TDT4290 - Customer driven project

ESUMSDroid:
Handheld Client for Heart Monitoring

Group 9:
Thomas Hoberg Burnett - Finn Robin Kåveland Hansen - Stian Veum Møllersen - Yushan Pan - Øystein Jaren Samuelsen - Dag Øyvind Tornes

Fall 2010

Summary

This report is a deliverable in the course TDT4290 at the Norwegian University of Science and Technology (NTNU) in the fall of 2010. It documents the process of developing ESUMSDroid, an application which runs on Android smartphones and serves as an important component in the Enhanced Sustained Use Monitoring System (ESUMS), which is a home health monitoring system, developed with the purpose of remotely monitoring the health state of patients with congestive heart failure. Currently it is aimed at health monitoring of US Military veterans. ESUMSDroid is the Android implementation of the ESUMS handheld client, which serves as a bridge between medical sensors used by the patients, and the ESUMS web server to which the helth data is sent over the Internet, and made accessible to health personnel. The handheld client also allows the patient to monitor his or her own health state via a graphical user interface. The customer who comissioned this software is SINTEF, the biggest independent research company in Scandinavia.

We would like to extend our gratitude to our supervisors, Sobah Abbas Petersen and Sundar Gopalakrishnan, and our SINTEF contacts Bjørn Magnus Mathisen and Anders Kofod-Petersen, who assisted us greatly in the development process.

Øystein Jaren Samuelsen

Thomas Hoberg Burnett

Finn Robin Kåveland Hansen

Stian Veum Møllersen

Yushan Pan

Dag Øyvind Tornes

Contents

List of Figures

List of Tables

Glossary

Activity An activity is an important component of an Android application. 35

ADT Android Development Tools, a plugin designed for Eclipse which provides the user with useful tools for developing Android applications. 34

apk A fileformat used in the Android operating system for installing bundled packages. 38

AUP Agile Unified Process, a software development methodology. 22

Bitbucket Bitbucket, a free and open source tracker. 26

chest unit A sensor unit which the patient wears around the chest. It measures various vital signs, such as heart rate, respiratory rate, activity level and posture. 2

COTS Commercial off the shelf. 21

Eclipse A java-based IDE, widely used for developing java programs. 37

ESUMS Enhanced Sustained Use Monitoring System. A prototype system developed for continuous monitoring of patients with congestive heart failure for the purpose of physiological rehabilitation. 1

FreeMpower Is an open platform of services that simplify and speed up the development of e-health services. 32

FTP Short for File Transfer Protocol, a protocol used to perform file transfer over the internet. 26

GUI Graphical User Interface. 25

handheld client The software that runs on the patient's smartphone, relaying measurement data from the chest unit and other sensors to the ESUMS web server, and allowing the patient to monitor his own measurements. 2

HDP Short for Health Device Profile, a Bluetooth communication profile. 39

HL-7 HL-7 is a standards organization which works to produce standards applicable to the field of health-care. 41

HTTP Hypertext Transfer Protocol. A network communication protocol. 32

IDE Short for Integrated Development Environment. 37

IEEE 11073 An IEEE standard detailing the communication between medical devices. 41

IRC Internet Relay Chat, a protocol for instant messaging over the internet. 29

IrDA Short for Infrared Data Association. 40

JVM Short for Java Virtual Machine, a program that is required to run java code. 38

LogCat A tool used for viewing debugging information when developing Android applications. 38

Mercurial Apache Subversion, a distributed version control system. 26

NONIN sensor A small sensor that clips onto a finger and measures oxygen saturation in the blood. Full name: Onyx II Model 9560 Bluetooth Fingertip Pulse Oximeter, produced by Nonin Medical. 2

SDK Software Development Kit. Consists of a set of development tools designed to ease the creation of applications for the platform the SDK is created for. 34

Service A service is an important component of an Android application. 35

SOA Service-Oriented Architecture. 32

SOAP Simple Object Access Protocol. An application layer protocol designed for the exchange of structured information in web services. 32

software architecture The structure of structures of a software system, which comprise software elements, the externally visible properties of those elements, and the relationships between them. 80

SPP Short for Serial Port Profile, a Bluetooth communication profile. 39

SVN Apache Subversion, a version control system. 25

TDD Test Driven Development, a software development methodology. 22

Trac Trac, an enhanced wiki and issue tracker. 25

UI User Interface. 66

View A view is an important component of an Android application. 35

Wi-Fi Common name for wireless technology implementing IEEE 802.11, such as wireless local area network (WLAN). 40

WSDL Web Services Description Language. An XML-based language used for modeling web services. 32

XML Short for Extensible Markup Language, a mark-up language created by W3C. 32

XP Extreme Programming, a software development methodology. 22

Yast An online hour logging service. 29

Chapter 1

Introduction

This chapter gives a brief introduction to the project. We give the background of the project and state the problem description. There is also an overview of the contents of the rest of the report.

Project name The name of the assignment is "Handheld client for heart monitoring". The name of our product will be ESUMSDroid".

The customer The customer is SINTEF ICT, represented by Anders Kofod-Petersen and Bjørn Magnus Mathisen. SINTEF is the largest independent research organization in Scandinavia. SINTEF is non-profit, and reinvests all their profits into new research. SINTEF has a partnership agreement with NTNU and shares personell and facilities with the university. Over the last couple of years, SINTEF has started to look towards the international market and is working with research initiatives within the EU. SINTEF ICT is the Information and Communication Technology department of SINTEF [5].

Stakeholders The stakeholders for the project include the customer (see the previous paragraph), the development team, the team's supervisors, the examiners who will be evaluating the project, and finally the users of the finished product, first and foremost comprising patients, but also nurses and doctors.

Project background and problem description ESUMS stands for Enhanced Sustained Use Monitoring System. It is a prototype system developed to enable continous monitoring of patients with congestive heart failure [4]. The goal of the system is essentially to allow these patients to live as normal a life as possible. In the past, they have had to make frequent trips to the hospital in order to let the doctors keep track of their health state. The ESUMS system will alleviate

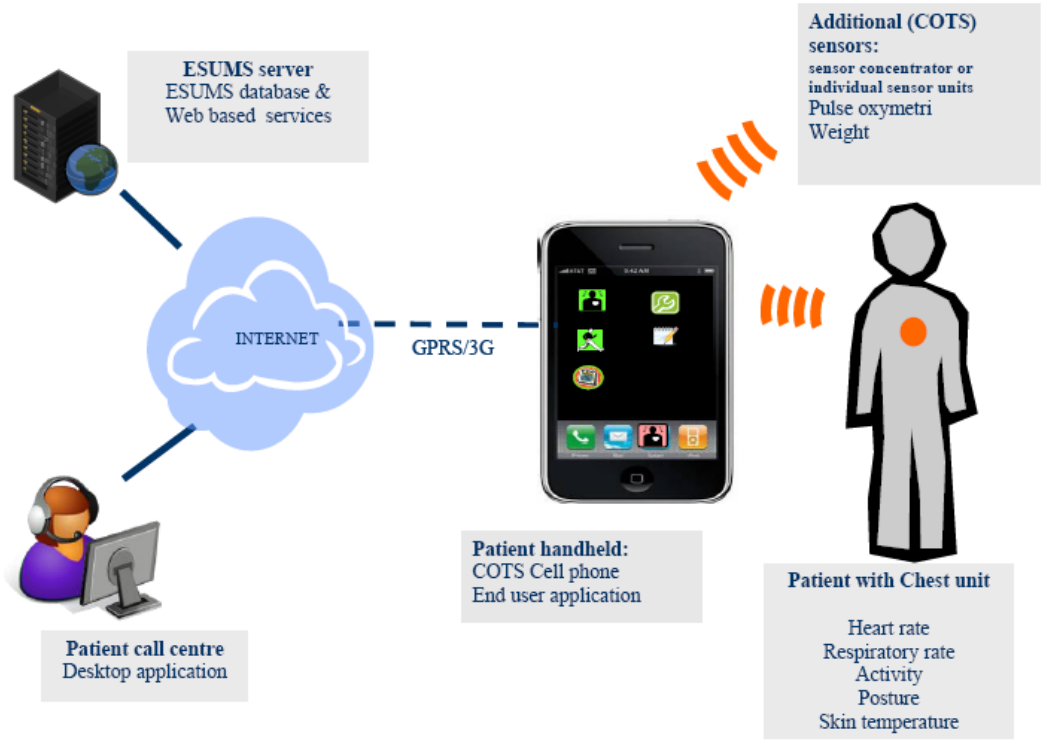

Figure 1.1: An overview of ESUMS [1]

this situation by allowing health personell to monitor the patients' heart function and other health related measurements remotely. An overview of the various components of ESUMS can be seen in figure 1.1.

The patient is given a belt, the chest unit, which monitors various attributes that describe the patient's current health state, including heart rate, respiratory rate, skin temperature, activity level, posture and some other measurements. The patient is required to wear the chest unit during the day. The unit sends it's measurements wirelessly via Bluetooth to the patient's smartphone, which in turn transmits the data to the ESUMS web server via Internet, allowing health personell to access it. In addition to the chest unit, the ESUMS system accomodates other external measurement devices that allow the patient to measure weight, blood pressure and possibly other data. At this point, only one such device - the NONIN sensor - is specified. It is a small sensor that clips onto a finger and measures oxygen saturation in the blood.

Of course, software is needed at every point in this chain of technology. This project is concerned with implementing the application that will run on the patient's smartphone, the handheld client. Its job is to request and receive data from

the chest unit and other external sensors, store the data until it can be transmitted to the ESUMS server (pending an available Internet connection), and finally transmit the data to the server. An incomplete version of the handheld client has been developed on the Windows Mobile platform, using the C# programming language. In this project, we are going to port the handheld client to the Android platform (version 2.x), using the Java programming language. This will also involve adding functionality that has not yet been implemented in the Windows Mobile version.

Measurement of project effects What the customer hopes to get from this project is a working prototype, which will prove that the ESUMS handheld client is feasible on the Android platform. A post-project goal is to have a working system for research purposes. A live test is going to be performed next autumn. However, it is unlikely that the Android version of the handheld client will be used in that test. (The Windows Mobile version will be used.) An even longer term goal is to develop the prototype into a quality system for industrial distribution. The hope is that several types of smartphone platforms can be used at this stage, including Windows, Android and possibly others as well.

Duration The project starts on 31 Aug 2010 and ends on 25 Nov 2010. This means we have a total of 12 weeks and 2 days to complete the project.

Report outline Chapter 1 (this chapter) provides a brief introduction to the project and the organization of the rest of the report. Chapter 2 contains the project plan, which states how the execution of the project will be managed and scheduled. In Chapter 3 we give an overview of various technologies and standards that are relevant to the project. We also provide a rationale for why certain tools and technologies were chosen over others. Chapter 4 contains the requirements specification for ESUMSDroid, including usage scenarios, use cases, functional requirements, the product backlog and non-functional requirements. Chapter 5 introduces the software architecture that we came up with before starting the actual Scrum development process. Chapters 6, 7 and 8 are dedicated to the three sprints. Each of these chapters contains a sprint plan, a sprint backlog, an overview of architectural, design, implementation and testing work that was done during the sprint, a summary of what was accomplished and a sprint retrospective. In Chapter 9 we evaluate our own work throughout the whole project, the quality of the final product, as well as the course itself.

Chapter 2

Project Plan

The purpose of the project plan is to state clearly how the project will be executed and managed. It is a dynamic document that is likely to be changed several times over the course of the project.

This chapter will consist of these sections:

- **2.1 Terms and resources:**
  This section describes the terms for the project, as well as the resources available to us.

- **2.2 Concrete project plan:**
  This section contains the concrete project plan, which discusses the actual schedule, the phases of the project and the workload.

- **2.3 Organization:**
  This section describes the organization of the team.

- **2.4 Rules, templates and standards:**
  This section is concerned with rules of conduct and templates and standards that should be followed in the work process.

- **2.5 Project management:**
  This section describes topics related to project management, such as documentation of project work, meetings and risk handling.

- **2.6 Quality assurance:**
  This section discusses the quality assurance routines of the project.

2.1 Terms and resources

This section describes describes the general terms for the project, set by the customer and the course responsibles, as well as the resources available to us during the project, including the time budget.

2.1.1 General terms - limitations and resources

The purpose of this project is to port an existing application to a different platform. This naturally puts certain constraints on our choice of tools and methods. We are going to port the ESUMS handheld client to the Android platform, using the Java programming language to implement it. The handheld client will be communicating with external Bluetooth sensors and with an ESUMS server running web services. The protocols for communication are beyond our control, so we will have to adhere to the protocol specifications we get from the customer. Unfortunately, the chest unit is currently being redesigned, so the communication protocol between chest unit and handheld client is not fully determined. This puts constraints on the early parts of development, as we will have to concentrate on aspects of the application other than the Bluetooth communication with the chest unit, until we receive a complete specification of the protocol from the customer.

All members of the development team must sign a non-disclosure agreement with the customer. See appendix A.4.

For testing the application during development, we will mainly be using an Android emulator (see section 3.4.2), but the customer will also provide us with Android phones, so that we can test the application in a more realistic environment. Some team members also have Android phones of their own. In addition we will be allowed to borrow a chest unit for a few days at a later stage in the project. Until then, data from the chest unit will have to be simulated when we perform tests that require this data to be available. For testing the interaction between the handheld client and the ESUMS server, we will install and run the server application on a desktop belonging to one of the team members. There are thin client computers available in the P15 building, which could be used for development. Unfortunately, running the Android emulator on these would be a test of patience at best. More powerful computers are needed for this. All team members have their own laptops (and desktops), which will be used instead. Unfortunately, Robin's laptop is old and quite unreliable, so he will have to spend most of the time at his desktop at home.

2.1.2 Time budget

We are to complete a large project in a relatively short period (three months), so we expect that we shall have to put a considerable portion of our available time into it. The course *Customer Driven Project* is worth 15 credits, which is equivalent to 24 hours of effort per week. Our team consists of six people, and the project will last 12 weeks and two days (ie. 12,29 weeks). This gives us a budget of 1770 hours. Note: The information booklet we were given in the the beginning of the project states that the total time budget should be 1872 hours for a six person team. This suggests a weekly effort of 25,4 hours per person per week. We see this as quite unrealistic, as we all take two subjects in parallell with this project, so we really cannot afford to spend more time on the project than what is expected based on the number of credits.

2.2 Concrete project plan

This section describes the concrete schedule for the project. We have decided to follow the Scrum methodology in this project (see section 3.1). Roughly the first four weeks of the project (starting 31 Aug) have been dedicated to planning, preliminary study, requirements specification and initial architecture work. After this, we will do three sprints, each of which will last two weeks. Each sprint starts with a sprint planning meeting and ends with a sprint review and retrospective. The first sprint will start on 27 Sep. The last sprint will end on 7 Nov. At this point, the application needs to be finished, ie. implemented, tested and documented. After the final sprint, we have set off two weeks to finalize our report and prepare for the final presentation. There is also a three day safety buffer after this before the presentation and delivery.

2.2.1 Phases

We have divided our project roughly into nine phases. The first four (Initial planning, preliminary study, requirements specification and initial architecture) are not clearly delimited, but rather overlapping in time. They could in fact be seen as one preparations phase, but we have chosen to separate them here, as they address different issues. The final two phases (project evaluation and report finalization/presentation) are also likely to be overlapping in time. The chapters in this report correspond roughly to the phases. The duration of each phase can be seen from the Gantt diagram in 2.1. Note that this diagram only shows the extent of each phase in time. Where the phases overlap, it says nothing about the amount of work budgeted for each phase. For details on workload per acivity, see section 2.2.2.

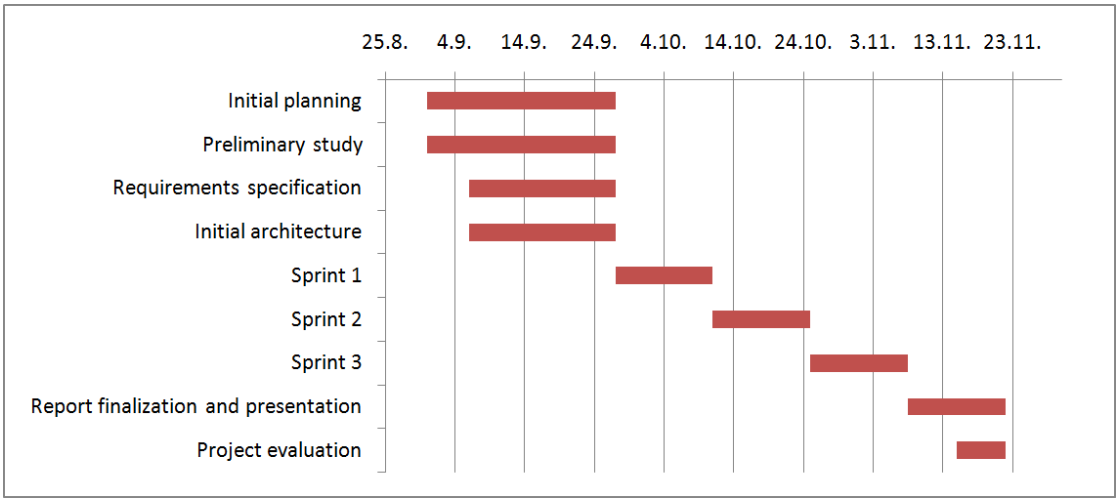

Figure 2.1: Gantt diagram showing the duration of each project phase

Initial planning

This is basically the first setup of the project plan. The project plan is of course updated and maintained throughout the project.

Preliminary study

The preliminary study involves research into various technologies and methods being considered for use in the project, so that qualified decisions can be made regarding what technologies and methods to choose. In our case, many such decisions have been made by the customer (ie. platform, communication protocols etc.), but many choices will still have to be made by the team. This includes choice of development method (Scrum, waterfall and others) and support tools, such as version control system, collaboration tools and file sharing aids. We will also look into some of the choices the customer has made, and find out whether they are feasible. In addition, the pre-study will be concerned with getting to know the technologies, protocols and standards that we are supposed to use.

Requirements specification

In many projects, this can be a very time consuming phase, often because the customer does not necessarily know exactly what they want. Requirements specification involves communicating with the customer to determine a set of requirements that the finished software system must satisfy. In the case of our project, the system that we are to build has already been developed (partially) on a different platform, and a more or less complete requirements specification exists. However,

some of these requirements are not final, and there may be some missing. Because of this, some modifications and additions to the requirements specification later in the project are likely. Additionally, most of the existing requirements are too high level to be of direct use in implementing the application, so we will have to break them down into more tangible tasks that will constitue the product backlog. This will partially be done in the requirements specification phase, and partially in the planning of individual sprints. To gain a better understanding of the application, we will also develop some activity scenarios and use cases for the application.

Initial architecture

Before the first sprint starts, we want to have the overall architecture for the handheld client pretty much determined, so that we can focus on detailed design and implementation already in the first sprint. There is of course a chance that the architecture may need revision later, but the handheld client is not a very complicated system with loads of functionality, so it should not be too difficult to come up with a feasible architecture. It has already been determined that a quite simple three layer architecture will serve us well.

Sprint 1

Sprint 1 will be about getting an initial implementation of our three-layer architecture up and running. For the sprint backlog, we will select the highest priority tasks that we can start right away. There are several requirements (mostly related to Bluetooth and we server communication) that are not yet fully determined, so these will have to wait. We *can* however implement the basic functionality of all three layers. (See chapter 5 for details on the software architecture.)

Sprint 2

The main concern of Sprint 2 will be to implement the communication between the handheld client and the external Bluetooth sensors, as well as the communication between the handheld client and the ESUMS web server.

Sprint 3

In the final sprint, we will put final touches on the functionality and make sure all parts of the application work seamlessly together. There is also the possibility of additional requirements from the customer at a late stage in the project. If there is time, we can try to implement them in this sprint.

Report finalization and presentation

In this phase we will finish the report and prepare for the final presentation of the project. The report is of course a continuous work in progress throughout the project, but it is very likely that we will need quite some time to put final touches on it and make sure it holds high quality. Therefore, we have set off a considerable amount of time for this. Some of this time could possibly be used for a short fourth sprint or an extension of sprint 3, should it turn out that we need a little more time for development and testing.

Project evaluation

In this phase we will evaluate the project as a whole, both in view of our own execution of the project, as well as the course in general. We will probably do most of the evaluation when the report nears completion.

2.2.2 Workload

We have a budget of 1728 hours for this project (see section 2.1.2). These hours are divided between ten different activities. Note that these activities do not correspond directly to the phases described in section 2.2.1. The phases are a division of the project into periods of time, although some periods may overlap. The activities are a division of the total workload. The activities are:

- **Project management** This is done continually throughout the project, but there will probably be more of it in the early parts. It includes setting up systems and tools, coordinating activities, attending meetings etc.
- **Planning** This is mostly initial project planning. (Some revision of the plan may be done later in the project.)
- **Study** It will be necessary for all team members to do quite a bit of studying to get up to speed on the technologies, tools and standards we will be using in this project. This includes preliminary study (see chapter 3), self study throughout the project, as well as attending seminars.
- **Requirements specification** This includes everything related to creating usage scenarios and use cases, specifying functional and non-functional requirements, and breaking the requirements down to tasks for the product backlog.
- **Sprint 1, Sprint 2 and Sprint 3** All work that goes into tasks from the sprint backlogs belongs here. It includes design, implementation, documenta-

Table 2.1: Workload per activity

Activity	**Planned time (hours)**	**Planned time (percentage)**
Project management	265:30	15.0 %
Study	354:00	20.0 %
Planning	88:30	5.0 %
Requirements specification	88:30	5.0 %
Sprint 1	88:30	5.0 %
Sprint 2	221:15	12.5 %
Sprint 3	221:15	12.5 %
Evaluation	88:30	5.0 %
Report writing	265:30	15.0 %
Presentation	88:30	5.0 %
Presentation	88:30	5.0 %
TOTAL	**1770:00**	**100,0 %**

tion (both commenting code and maintaining architecture and design/implementation documents) and testing. Sprint planning meetings, scrum meetings and sprint review and retrospective meetings also belong here.

- **Evaluation** This is the time that goes into the project evaluation phase at the end of the project.

- **Report writing** A considerable amount of time will be spent writing the project report. This will go on throughout the whole project.

- **Presentation** This includes all time spent preparing for the final presentation.

The planned division of workload between the various activities is given in table 2.1. We came up with these numbers by looking at the time spent by other groups on the various activities in past projects, and we adjusted the numbers slightly to fit our project. For instance, we expect the time spent doing requirements specification to be quite low, due to the fact that we have a more or less complete requirements specification provided by the customer. The time estimate for sprint 1 is lower than the other two sprints. This is due to the fact that we are lacking important documentation on communication protocols at this early stage, which means that the amount of development work that can be done in sprint 1 is quite limited.

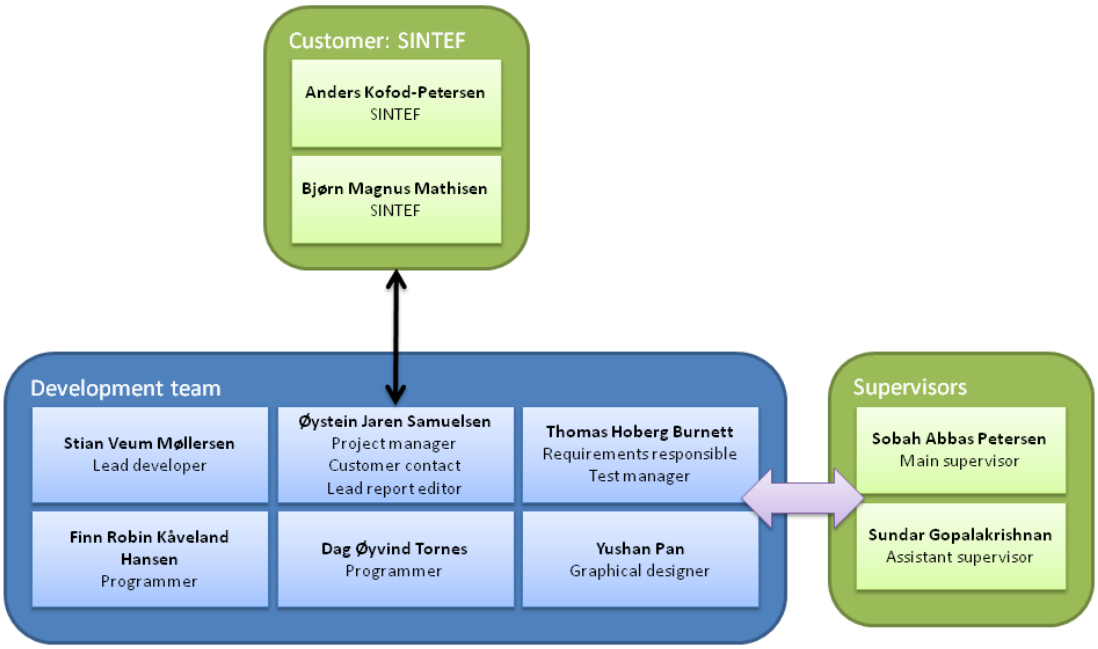

Figure 2.2: Organizational chart for the project

2.3 Organization

Using Scrum as development method typically suggests a quite flat organization structure. Instead of having well defined roles dictating the strict responsibilities of all the team members, the tasks that need to be done are allocated dynamically to the team members at the beginning of each sprint, and sometimes also during sprints. Even so, we figure it is a good idea to allocate certain key responsibilities in order to ensure the quality of the various aspects of the project. However, instead of allocating all such responsibilities immediataly, we have decided to experiment for a little while to see what the various team members enjoy and excel at. The result can be seen in figure 2.2.

In addition to their key responsibilities, everyone will be contributing to development and writing to a certain extent.

2.4 Rules, templates and standards

This section describes rules, templates and standards that should be followed in the work process.

2.4.1 Language

The language used in all code and delivered material must be English. All communication within the team, intended for all team members, should also be in English.

2.4.2 Document templates

Templates must be used when writing meeting agendas, minutes and weekly status reports. We will create LaTeX templates for these documents. Examples of such documents are included in appendix A.

2.4.3 Use of version control system

The following rules apply when using the version control system:

- Always make sure (to the best possible extent) that your code is bugfree before pushing to the repository.
- Commit whenever you have made a noteable change/addition. Write a brief commit message that effectively and clearly states what you did. If possible, don't include many changes in the same commit. Commit several times instead.
- Always pull before you start working on something. Push at least when you are done working for the day, preferably more often. (You can even push after every commit.)
- No compiled files or auxiliary files should ever be added to the repository. Ie. the repository should only contain source.

2.4.4 Use of DropBox

- Never work inside the DropBox folder, as this leads to lots of useless files being synchronized (such as the auxiliary files created when building latex files).

- The dropbox folder is for documents that are mostly finished work (like minutes, agendas, weekly reports, images etc). When you finish work on a document you want to share with the team, store the finished document (a pdf for a weekly report for instance) in the approperiate folder and the raw (the .tex file for instance) in the raw folder located the same place as the finished file.

2.4.5 Coding conventions

- Coding style should be in accordance with Checkstyle (see section 3.4.2). Some deviations are allowed, such as short variable names (i, j, etc.) for loop counters.

- All externally visible (public) classes and methods must be documented with Javadoc.

- Use inline comments to explain your code where necessary, especially if you do something slightly strange or unconventional.

2.5 Project management

This section describes how the various management aspects of the project will be handled, including documentation of project work, regular meetings and risk handling.

2.5.1 Documentation of project work

We have created a shared Google spreadsheet for logging of all time spent on the project. The spreadsheet contains one sheet for every week of the project, and every week has a form for each team member. Time is logged per activity and (where applicable) per task ID (tasks from the sprint backlog). This allows us to create simple status reports every week. The sprint backlogs are also managed in this spreadsheet. These include a sprint burndown table (and chart) for each sprint, which is updated at the end of every day, allowing us to keep track of progress through the sprints.

2.5.2 Regular meetings

This section describes the various types of regular meetings that will be held throughout the project.

Customer meetings

We will have a meeting with the customer approximately every other week. This will usually be on Mondays, in the period 08:15-10:00. We would have preferred a little more frequent meetings, but regrettably this is not possible, as the customer representatives are very busy. They are however available on e-mail, so we will be able to ask questions outside of meetings. The meetings with the customer will be used to clear up any misunderstandings that may have occured and get answers to questions we may have that are difficult to discuss via e-mail. The meetings will also be used to show the customer how we are doing and get feedback. We will try to make the customer meetings coincide with sprint planning meetings if possible, as the customer should be involved in the prioritization of requirements.

Advisory meetings

We will meet with the supervisors every Tuesday at 08:45-09:45 to get feedback on our work. Issues related to work process, project management and the writing of the report will be discussed here.

Sprint planning meetings

The project will consist of three sprints, and each one will be started with a sprint planning meeting. In these meetings, we will formulate a goal for the sprint and prioritize and select requirements from the product backlog to be implemented in the sprint. The sprints each last two weeks, from Monday to Sunday, so the sprint planning meetings will be held every other Monday (08:15-10:00) while we are doing the sprints.

Sprint review meetings

These meetings involve demonstrating the results of the sprint to the customer and getting feedback on what we have accomplished. We may not be able to do this properly, ie. at the end of the sprints, as the customer representatives do not have time for more than one meeting every other week. Because of this, we will simply have to include some demostration in the customer meetings we do have.

Sprint retrospective meetings

These meetings are held at the end of every sprint to evaluate our own work during the sprint and come up with ideas for improvement to the work process. We will try to fit this in at the end of the last week of the sprints. It should not be a long

meeting; only 15-20 minutes. The result of the sprint retrospective meetings will be included in the last section of each sprint chapter (sprint retrospective).

"Daily" scrum meetings

Ideally, we ought to do a quick scrum meeting at the beginning of every day, so that everyone is updated on what each team member has worked on, what he has been having difficulties with, and what he intends to work on next. Even though this should only take 10-15 minutes, it is not practical to do this every day, as we all take other subjects besides this project, and there are relatively few time slots where all the team members are available at once. We will have scrum meetings on Tuesday mornings, just before the advisory meeting, on Thursday mornings at 10:15, and sometimes also on Monday mornings at 08:15. In a way, this does make more sense than having daily scrum meetings, as all the team members take two subjects in addition to the project, so the project constitutes only 50 % of the total workload. In light of this, a scrum meeting approximately every other day seems appropriate.

Other team meetings

If there is something in particular that needs to be discussed with all the team members, we can set up a meeting on Wednesdays in the time period 12:15-16:00. Mondays (08:15-10:00) is also a possibility in the weeks that do not have a customer meeting.

2.5.3 Risks

We have identified a set of possible risks that may cause trouble for the team in executing the project. Each risk has an associated probability (how likely it is that the risk will take effect) of low (L), medium (M) or high (H), as well as a measure of consequences (how severe the consequences would be if the risk was to take effect), also of low (L), medium (M) and high (H). For each risk, we have come up with a strategy for how it shold be avoided, or alternatively dealt with if it takes effect. An overview of risks and strategies is given in table 2.2.

Table 2.2: Risk handling

ID	Risk factor	Prob.	Cons.	Strategy
1	Team members getting sick	H	M	Reduce consequences: Plan so that there is time to spare. Make sure several people know how everything works, so that tasks can be easily transferred between team members
2	Communication protocols unknown/subject to change	H	M	Reduce consequences: Plan so that work that does not depend on the communication protocols can be done early on in the project.
3	Communication/ language problems lead to misunderstandings in the team	H	L	Avoid: Keep in mind to communicate clearly. Always speak English in team discussions.
4	Misunderstandings between team and customer	M	H	Avoid: Use meeting minutes to get confirmation on decisions/points discussed. Contact customer with questions when uncertainties arise.
5	Our product does not fill the customer's needs/is not what the customer wanted	L	H	Avoid: Involve customer in the development process as much as possible (prioritizing requirements, sprint reviews).
6	COTS (ie. BitBucket) no longer free	L	L	Deal with it: Pay for usage. (It will not be expensive at any rate.)
7	Mismatch in team members' schedules cause collaboration problems	M	M	Reduce consequences: Plan work so that team members can work independently. Use available common time for planning.
8	Product impossible to implement satisfactorily (due to unforeseen technical contraints)	L	H	Avoid: Do a proper preliminary study to gain confidence that the product can be implemented satisfactorily.

Continued on next page

Table 2.2 – continued from previous page

ID	Risk factor	Prob.	Cons.	Strategy
9	Schedule overruns	M	M	Avoid: Plan so that there is a comfortable time buffer. Work steadily throughout the whole project. Backup plan: Work extra hard towards the end if necessary.
10	Fighting within the team	L	M	Avoid: Discuss problems with each other. If need be: Take it up in team meetings and/or discuss with supervisors.
11	Customer introduces additional requirements	L	L	Deal with it: Try to implement towards the end of the project if time. (We have been assured that this will not be critical.)
12	Someone messes up the code, so that work is lost	L	M	Avoid: Utilize a version control system and set clear guidelines for how it should be used by team members.

2.6 Quality assurance

This section outlines how we intend to ensure the quality of the delivered product and all written material that will be delivered.

2.6.1 Routines regarding the customer

We will have a meeting with the customer approximately every other week. These will usally take place on Mondays, starting at 08:15. Callings for customer meetings must be sent to the customer by e-mail at least 48 hours before the meeting. The calling must include the time and place, the purpose and the agenda for the meeting. It must also state what preparations the customer should make before the meeting. At the beginning of every customer meeting, one team member will be appointed to write minutes. These must be sent to the customer (and to the team's mailing list) by the end of the day of the meeting. The customer will respond within 48 hours and approve the minutes, or provide corrections if necessary. Questions sent by e-mail should also be answered by the customer within 48 hours. All e-mail correspondance should be sent to both customer representatives, as well as the team's mailing list.

2.6.2 Routines regarding the supervisors

We will have advisory meetings with the supervisors every Tuesday, usually at 08:45. The calling for the meeting should be sent by e-mail to the supervisors by 14:00 the day before, so that the supervisors have time to review our material for feedback. The calling should include time and place, meeting agenda, a weekly status report (including hour log for the last week), minutes from last week's advisory meeting, as well as minutes from other meetings held during the last week where important decisions were made, the current edition of the report and a changelog for the report. The subject field of the e-mail must contain "Kpro9". At the beginning of the meeting, one person is appointed to write minutes. The minutes are made available to the team (in the DropBox) by the end of the day, and are included in the calling for the next advisory meeting.

2.6.3 Internal routines

It is very important to reduce the risk of misunderstandings within the team. That risk is especially high in our team, since we have a team member who does not speak Norwegian, and also has limited experience with English. This means that we will have to keep in mind to communicate clearly and ask extra questions to make sure that no misunderstandings occur. It is also important that we remember to speak English in team discussions.

When working on the project, it would be preferable to have all team members in the same room as much as possible. This turns out to be challenging, however, as the various team members have very different schedules, so there is a very limited amount of time available where all the team members can meet at once. Furthermore, we need some decent computers to do the programming work (the Android emulator is quite heavy on the system resources), and we need to be able to install whatever software we want. In this light, the thin clients at P15 are simply not cut out for the task. We all have laptops, but unfortunately, they are not all in very good shape, so some team members will have to do most of the work on their home desktops. Consequently, we will have to communicate regularly, both via the IRC channel that we set up for this project (see section 3.2.3), and via the team's mailing list. Everyone should check these communication channels at least twice every day, and always stay logged in on IRC while working.

2.6.4 Testing

All code produced during the project must be tested before delivery. There are mainly five forms of testing that will be performed:

- Unit testing: This is testing on the class/method level, and it will mostly be done concurrently with programming.
- Integration testing: These tests are meant to verify that the interfaces between modules work as specified, ie. that the various modules of the system function together as they should.
- System testing: Testing the whole system to make sure it meets the requirements.
- System integration testing: Making sure the system works correctly with external systems. In our case: Making sure the handheld client functions correctly with external Bluetooth sensors and with the ESUMS web server. We expect that this will be done in combination with regular system testing (previous bullet point), as system testing without the external systems in our case will be quite pointless, because all the functionality in the handheld depends heavily on external systems.
- Acceptance testing: This is usually performed by or with the customer. The purpose is to decide whether the system is built to satisfaction, ie. accepted or not. In our case, we have agreed with the customer that the acceptance testing will be based strictly on the functional requirements described in section 4.1.3. This means that acceptance testing will be equivalent with system testing in this project.

Chapter 3

Preliminary Study

This chapter is intended to give an insight into the process which took place before any code was written. We spent a significant amount of time researching the technologies we were required to use, and the technologies we were able to choose ourselves. Many of the technologies that were to be used had already been selected by the customer, but there was some freedom around others, and in this case especially the collaboration tools used by the group, which we discuss at some length in this chapter.

While the platform, and some communication protocols had already been selected for us, we were on our own in deciding on how exactly to use these, and which parts of them to use. These technologies are discussed in this section, but the specifics of how they were used in the codebase in the end have naturally been left out.

During our first meeting with the customer, we were given a detailed description of the complete ESUMS project, and some of the specifics of our particular piece of it, namely the Handheld Client. In this meeting some key elements were laid out, and these were used to guide all decisions to be made during the preliminary studies, and also at later stages of the project. These elements were identified:

- **Uptime requirements**
  The ESUMS project as a whole is a class 5 system, meaning it needs 99.999% uptime. This is a requirement of nearly all medical equipment to be deployed in the US, which is one of the target markets of the ESUMS system. This means the system can have a total of 5 minutes downtime per year. This requirement is for the system as a whole, not for each part individually. Obviously, such a requirement will be a factor in any decision made.

- **Extensibility**
  One of the goals of the ESUMS project is to be a complete health data monitoring system. As one piece of hardware would be hard pressed to

measure all possible health data, one very much desired piece of the puzzle is extensibility. The Handheld Application should be able to read data from a variety of COTS (Commercial off the shelf) sensors. While each such sensor would require a bit of new code in the application, this code should be kept to a minimum, while the architecture should be designed with extensibility in mind.

- **Standards compliance**
  Medical systems are required to follow several standards, for interoperability, availability and performance. These systems may be crucial in a life-and-death situation, and as such are held to high standards. Computerized medical systems are still a developing field, but some standards are already in place, and several more are being developed. It is desired that the ESUMS system should be as compliant with these standards as possible.

This chapter will consists of these sections:

- **3.1 Software development methodology:**
  Analysis of different software development methodologies that the group can use to ensure a good development process.
- **3.2 Collaboration and communication tools:**
  Analysis of different tools for collaboration and communication inside the group.
- **3.3 Existing systems:** Analysis of the existing ESUMS and the components ESUMSDroid will interact with.
- **3.4 Technology for ESUMSDroid:**
  Analysis of the technology used to develop and run the ESUMSDroid application.
- **3.5 Standards:**
  Analysis of the different standards used by ESUMS and ESUMSDroid.
- **3.6 Preliminary study conclusions:**
  The conclusions made during the preliminary study.

3.1 Software development methodology

For a long time the waterfall method was the only prevailing methodology for developing software, however in the few recent years several new agile methods have been gaining ground in the field of software development methodologies.

When choosing such a methodology several factors must be considered. Some, for instance, set strict limits on roles or timelines. Others require involvement by external factors. Our decision finally fell on Scrum, but not without debate or thorough consideration. Before choosing, we tried to identify our choices. While several options were identified, such as XP, AUP, TDD and others, two were familiar enough to be analysed more thoroughly: Waterfall and Scrum.

3.1.1 The waterfall method

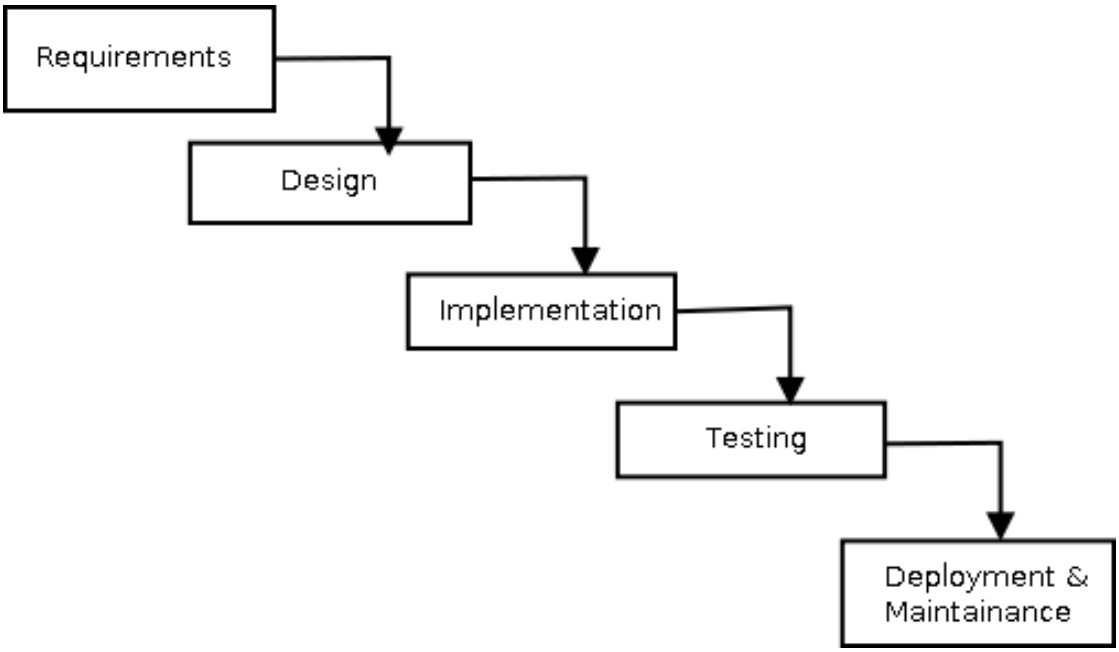

Figure 3.1: Illustration of the waterfall method for software development.

The waterfall method is extremely simple, and therefore easily implementable. It works by dividing up the development process into phases with dependencies, meaning that you don't move to the next phase before finishing the previous phase. Figure 3.1 shows a typical example of the waterfall method. The waterfall method also offered a clear path ahead, which had a certain allure to a team of mostly shotgun coders. It was also felt that with the short time we had available a more agile methodology might not show its true potential, and that the waterfall method might be adapted to fit our needs.

The main problem we found in waterfall was its rigidity. All parts of the project plan are laid out in a linear sequence, with a strict timeline governing it. We were concerned this rigidity might leave us precious little time at the end of the project, should we encounter any delays or unforeseen problems. This was

especially concerning given the lack of complete specification of the systems which ESUMSDroid had to cooperate with, going with the waterfall method would leave us with very little wriggle space in case of a change in the specification.

3.1.2 Scrum

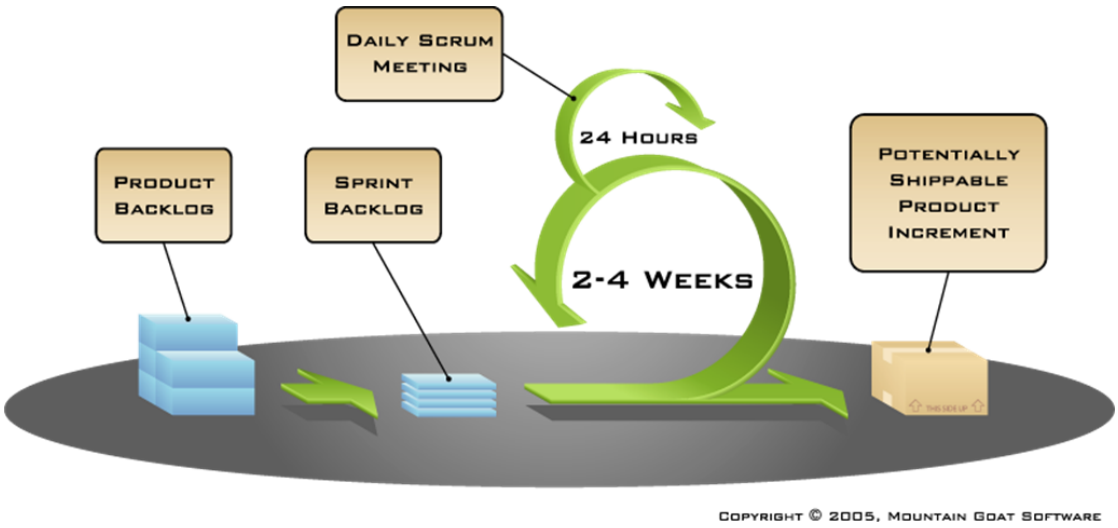

Figure 3.2: Scrum development method [2].

Unlike waterfall Scrum is an iterative process, where you complete a full cycle of development[1] each iteration. The illustration in figure 3.2 shows a nice overview of the scrum process. Each iteration in the process is called a sprint and is typically limited to a certain time frame (two to four weeks is the norm [6]). During a sprint the team will have short but frequent meetings called *Daily scrum*, where each team member presents the work done since last meeting, what they plan do to before next meeting and if they had any problems that prevented them from accomplishing the planned work [6].

The scrum process has a set of predefined roles [6] which are Srcum Master, Product Owner and the previously mentioned Team. The scrum master is the person responsible for maintaining the process and is analogous to a typical project manager role. The product owner is usually a person representing the stakeholders in the project. The team consists of the developers of the system.

In addition to the daily scrum mentioned, the scrum process includes a set of meetings which are Sprint Planning Meeting, Sprint Review Meeting and Sprint Retrospective[6]. The main function of the sprint planning meeting is to select

[1] requirement specification, design, implementation, testing, deployment

which requirements that will be implemented during the sprint and estimate time needed to complete each tasks. The review and retrospective meetings aim to identify problems and areas of improvement for next sprint and to summarize the work done in the current sprint. The review meeting also presents an opportunity to inform the stakeholders of the progress made and to demonstrate the current state of the system. A tool worth mentioned is the burndown chart. The burndown chart is used during the sprint retrospective (you can find an example of it in 6). The purpose of the burndown chart is to graph the progress of the sprint as summarized estimated time remaining of all the backlog items for each day.

In the scrum process you divide the requirements into smaller tasks called the Product Backlog, of which each Sprint Backlog is a subset of[6]. In addition to the backlogs you have a chart displaying the progress of each backlog item as the sprint goes.

3.1.3 Choice of methodology - discussion

We were sure from the start that we wanted a flexible methodology, as our problem domain was not fully understood yet. We knew requirements might change, our platform was mostly unknown to us, and we had a new and inexperienced team. With these difficulties clear to us we were sure we needed an approach which let us adapt as quickly as possible when difficulties arose. Scrum, with its short sprints and active encouragement of frequent communication seemed to fit this bill.

Scrum is of course not without its problems, as we soon realized. Our biggest problem in this project was Scrums stringent requirement that the customer be highly involved in the process, with a short feedback loop, and the need to manage a product backlog which the team can use to plan sprints. We knew from the start that the customer was very busy, and that this portion of Scrum would suffer for it. Still, we believed this would not be an insurmountable problem. Another problem we faced in choosing Scrum was the lack of a Scrum master. This is a person intimately familiar with the Scrum process, which keeps the team on track, where process is concerned. Our group contained no such individual. Still, having no one with strong feelings about the topic allowed us to adapt the methodology in ways to fit the team better.

Another issue is the formulation of the items in the product backlog (ie. requirements). User stories are a widely used approach in this respect. Elaborate information on user stories can be found in [7]. A user story describes some function that the software system should have, from the viewpoint of the user who desires this functionality. A user story is typically phrased like this: "As a <type of user>, I want <some goal> so that <some reason>". User stories can be defined on many levels of complexity, with high level user stories being split into several lower level user stories, forming a hierarchy that describes the system's

desired functionality from a high to a low level. User stories also present a simple and intuitive way to convey the requirements to a customer with little technical knowledge.

Even though user stories is the natural "Scrum way", we see some problems with this approach. A user story should describe a piece of functionality that actually does something useful for a user. Unfortunately, we are lacking information about the communication protocol with external Bluetooth units, and with the ESUMS server[2]. This means that in the first sprint, we will only be able to implement the parts of the system that do not directly involve communication[3]. With a few minor exceptions, no useful user stories can be defined without involving such communication. Furthermore, we already have a requirements specification that was provided by the customer (see section 4.1.3, and the customer representatives are both technically adept, so the argument of intuitiveness of the user story approach becomes less important. For these reasons, we have decided not to utilize user stories. Instead, our product backlog will consist of concrete tasks that together implement the requirements specification. The full product backlog, and further explanation of it's structure, is given in section 4.1.4.

3.2 Collaboration and communication tools

This section describes various tools that can be used to facilitate collaboration and communication between team members in a software development project.

3.2.1 Version control system (VCS)

Version control systems are a vital part of any software development project, and this was fully understood by the team. As such we wished to make as fully an informed decision as possible when making our choice. All members of the team had experience using SVN, but some had progressed to greener pastures, namely distributed version control systems, and fought valiantly to get these accepted.

When choosing a VCS, one is not only choosing a software package, but an entire support system for the development process. The course responsible suggested using Trac, and being familiar with it we wholeheartedly agreed. This decision was not without its problems however. We were ever so politely informed that configuring a new trac on any of the university's servers would take 2-3 weeks minimum. This was unacceptable to us, so we got hold of a private server and undertook the

[2]The customer will provide the necessary documentation eventually, but it is not available at this time

[3]In some cases, such as in the GUI, we can implement generation of dummy data to simulate the data from external Bluetooth units

task ourselves. Yet again we hit a brick wall, and were given given new insights in the levels of frustration which exists as we tried to configure our newly created trac (We also learned why it would take several weeks to configure.). Another issue was that we would have to take regular backups of the system.

We still had options in commercially available systems, but were not certain this would work with the need for secrecy required by the customer. This had in fact been our first choice, but was initially rejected for this reason. Seeing no other options, we approached the customer with the suggestion, and were thankfully given the go-ahead.

Our choice fell on Bitbucket[8], a free and open source tracker, with tight integration to Mercurial[9]. Picking bitbucket before one of the many other solutions out there was fairly easy. First, it offered private repositories, which would let us keep most of the requirements of the customer. Second, mercurial[9] is a fully featured distributed version control system available for all major platforms, including integration in our chosen IDE, Eclipse. And third, bitbucket provides a neat backup feature for us, which means that we would not have to worry about that.

3.2.2 File sharing

We chose to have an additional way of sharing files inside the group, this was decided based on two reasons. One, because VCSs don't work well with binary files (Such as PDF, doc and others) due to the inability to do merging and version control in other ways than simply replacing the entire file. And two, because of the sheer amount of documents generated throughout the course of the project (weekly reports, minutes, agendas etc.), which would make maintaining a neat repository in Mercurial difficult.

Because many of the members of the group had good experiences with a system called DropBox[10], the choice was easy. DropBox offers easy integration into the most widely used operating systems (Windows, Mac OSX, Linux) and a selection of mobile devices (Android and Mac iOS) and easy sharing of folders and files between group members. An example of DropBox integration with Windows explorer can be found in figure 3.4. DropBox also has features that prevents synchronization mishaps (where one user edits a document that another person is trying to sync) and an easy way of undoing deletions and otherwise fatal mishaps related to information going missing and best of all, it is fully automated.

Because of the previously mentioned experience with the system there was never really any other options up for discussion, but we can mention a few alternatives to illustrate why DropBox is good. An FTP server could have been used, but this would require all group members to install and use an FTP client, instead of using the fully automated DropBox client which is integrated into the operating

Figure 3.3: A screenshot of some commits in the repository.

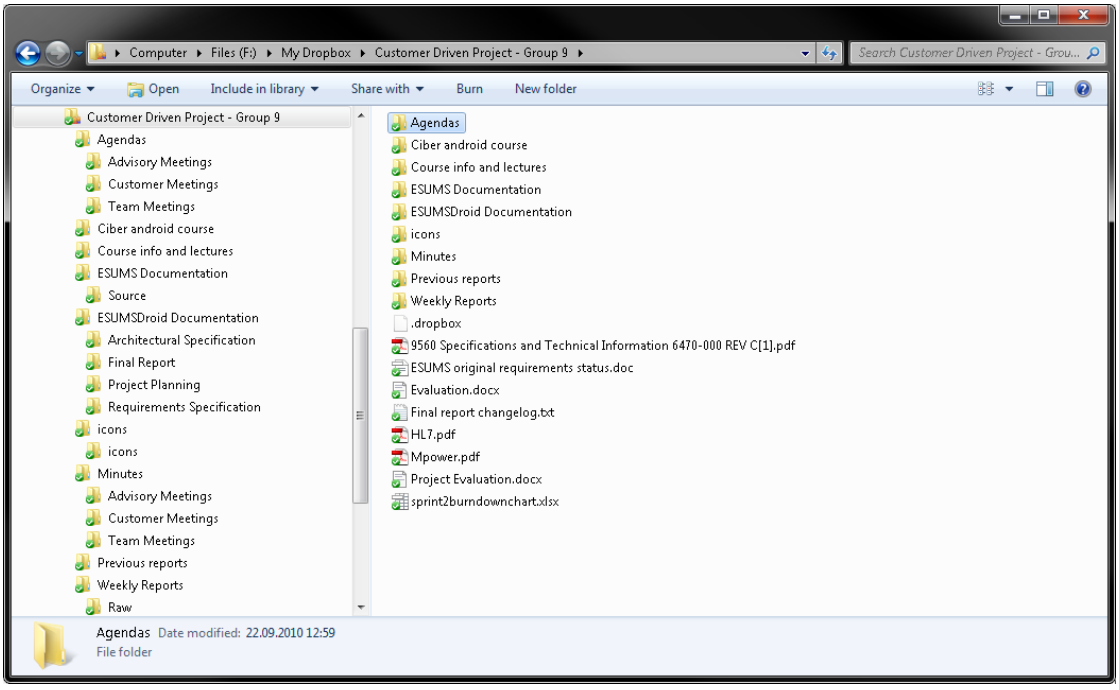

Figure 3.4: A screenshot of the dropbox folder in Windows.

systems native file explorer. This would also require someone to host said FTP server. Sharing via email is another alternative, but this gets messy very fast and requires significant overhead in organization per group member. Neither of these two systems offer any way of ensuring synchronization or a simple version control.

3.2.3 Remote communication

Email

One of the first things we did regarding group organization was to create a mailing list to use for distribution of important messages and other communication. Email as the de facto tool for group communication has been well established in today's digital everyday, so this felt almost as natural as breathing for the members of the group. The reason for creating our own mailing list was to be able to moderate it ourselves, as not all the group members use their @stud.ntnu.no-address as their main email. Also, that the mailing lists provided by the course staff did not appear until almost a month into the course.

Other

Because of a high mismatch of schedules among the members of the group, most of the work done would be done individually. This created a need for a different channel of communication other than email. This is mainly because maintaining huge email-threads is tedious and require some overhead. To solve this problem we decided to us an instant messaging service. The instant messaging service would need to be usable by multiple people on multiple computer simultaneously, be easily accessible, be available on all platforms and have options for saving conversation histories.

One system that fits all of those requirements is IRC, which is a protocol that have been implemented in a multitude of programs and runs on several different servers across the internet. Because of the ease of setting it up and maintaining it, and good experiences from previous group project by some of the group members this was chosen as the secondary communication channel (after email). Many IRC clients (a program that have implemented the IRC protocol) also support persistent connections, and one of these (irssi) is readily available on the stud-servers of NTNU. These things made the use of IRC an easy choice.

3.2.4 Hour logging and sprint progress tracking

Because of the nature of the project we would have to log hours spent during the entire project. We decided early on that this would require a tool with certain qualities. First, it would have to be accessible to everyone in the group and have the ability for everyone to edit at the same time. Second, it would have to be easy to sum the efforts of all group members and to transfer those numbers to weekly reports. Third, it would have to be extensible and easily manageable.

Based on a recommendation by some group members we decided to go with Yast[11], which is a system for logging hours in the browser using some nifty user interface with drag and drop mechanisms and the ability to export to text(HTML) and pdf formats. The downside, as we discovered fast, was that every member had to set this up for themselves and everyone had to manage the categories and make sure everyone was using the same format. And missing an easy way to sum numbers of all members created significant overhead in writing the weekly reports.

A new way of logging hours was sorely needed, and additional alternatives were considered. The extensibility and management aspect was prioritized so some sort of shared spreadsheet would solve that neatly. A shared spreadsheet over dropbox could lead to some clutter with simultaneous editing, an issue that would also exist with passing it around over email. Putting the spreadsheet into the VCS repository would also lead to more overhead than necessary. In the end we saw that the obvious solution would be to use Google docs[12], an office-like system

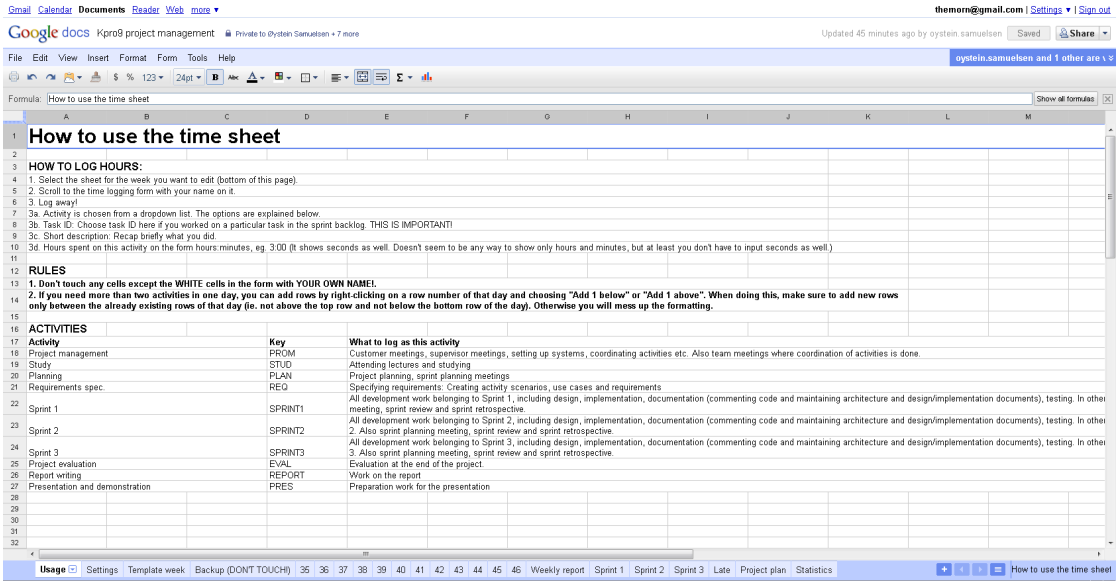

Figure 3.5: A screenshot of the Google Docs spreadsheet.

hosted by Google and used through your browser. This would enable us to share the spreadsheet easily and make it very easy to manage and export numbers to weekly reports. A screenshot of the spreadsheet can be found in figure 3.5.

After our planning had resulted in the choice of SCRUM as our development methodology, the choice of tool to track sprint progress felt easy as we already had the hourlogging system set up in Google docs. As tracking sprint progress is a task very similar to tracking hours spent on various aspects of project work, this was easily added to the spreadsheet and integrated with our initial hourlogging system.

3.3 Existing systems

This section will describe the existing systems in the ESUMS in more detail, and especially the parts that our application is going to be interacting with. An overview of the current ESUMS system can be found in figure 3.6.

3.3.1 ESUMS Server

The ESUMS infrastructure is designed around a client-server architecture. The ESUMSDroid application functions as the client. The server is needed to store data collected by the client, as well as make this data available to medical personnel. Therefore strict requirements regarding scalability and reliability were enforced by

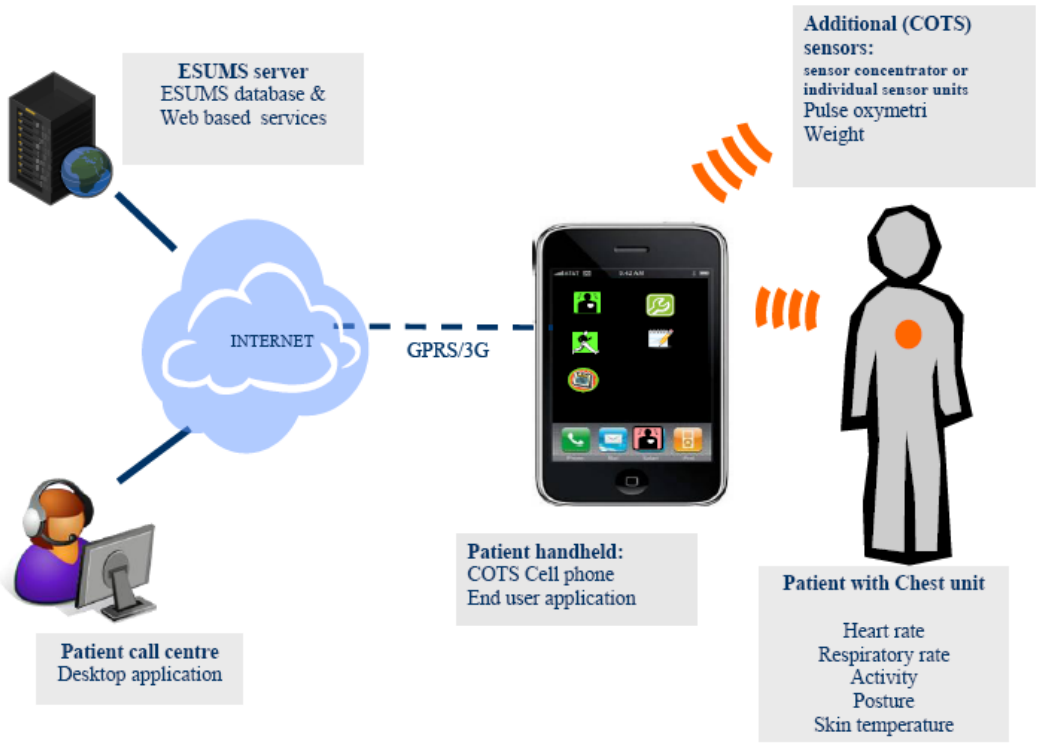

Figure 3.6: An overview of the current ESUMS, found in [1]

SINTEF during their implementation. This is beyond the scope of the project however, and will not be detailed in this report. See [1] for details. This section focuses mainly on the technologies used to implement the server, for the purpose of creating a compatible client application.

FreeMpower

The ESUMS system is built around the MPOWER framework. MPOWER is an open platform designed to make development of services for people with cognitive disabilities and the elderly a more efficient task[13]. The server uses a service-oriented architecture (SOA) based on FreeMpower. The main MPOWER services incorporated into the ESUMS relevant for this project are the User Medical Data Management service and the Security services. These two sets of services have been modelled using WSDL[14]. WSDL is described later in the chapter. Communication between the services and clients is done using SOAP[15] over HTTP.

Communicating with the security services is the first step in the information exchange between the server and ESUMSDroid. These services are used to authenticate the client and handle access control and permissions. Authenticated clients recieve a security token managing permissions. After the initial hand shaking, all transmissions from the client must include the security token for the server to accept any data.

The User Medical Data Management service is the part of the system which, among other things, handles the storing of sensor data. In other words, this is the service that ultimately receives most of the information exchanged with ESUMSDroid. When ESUMSDroid sends datapoints to the server, the transmissions must follow the format defined by this service. The service handles the proper storing of data points recieved. Data storing is done using an Oracle database[1].

WSDL

The Web Services Description Language (WSDL) is, as the name implies, a language created to clearly define and model web services. It is based on XML. It defines methods used for communication and the format of any objects used by these methods[14]. It also maps out objects, data types, services and bindings with the transport protocol, among other things. The MPOWER framework describes web services that are implemented using SOAP by providing WSDL definitions.

SOAP

SOAP is the protocol used in the WSDL description of the ESUMS web service. SOAP, or Simple Object Access Protocol, is "a lightweight protocol intended for exchange of structured information in a decentralized, distributed environment"[16].

Basically, it is the protocol ESUMS uses to exchange messages between the server and handheld clients. ESUMSDroid must therefore serialize the datapoints it collects into SOAP-compatible objects before transmitting them. Responses from the server obviously also follow this message format. Incoming SOAP objects therefore need to be deserialized into corresponding java objects to be able to update the internal status of ESUMSDroid. One problem with this though, is that officially, Android does not support SOAP. No official Android SOAP library currently exists. However, an implementation of kSOAP2[17] has been tailored towards Android. kSOAP is a SOAP library designed specifically for mobile Java environments. Therefore this makes it an obvious choice for our project.

3.3.2 Sensors

One of the main goals for ESUMS is to communicate with sensors to retrieve various medical data. The handheld client will function as a sort of bridge between a local sensor and a remote server as illustrated in figure 3.6. The main sensor device for the ESUMS is the chest unit, it is a sensor unit which the patient wears around the chest. The chest unit measures various vital signs, such as heart rate, respiratory rate, activity level and posture and communicates through the bluetooth protocol. At the time of writing the customer introduced us to another COTS sensor, the Nonin Onyx II. The Nonin sensor is a small sensor that clips onto a finger and measures oxygen saturation in the blood and, like the chest unit, communicates through the bluetooth protocol.

The goal for ESUMSDroid is to support both these sensors and make it easy to add additional COTS sensor devices.

3.3.3 Existing Windows Mobile prototype

Prior to our project, the customer had already created a demo of the ESUMS project. Part of this was developing a proof-of-concept handheld application. We were given access to this application, which was developed for Windows Mobile in C#. A few lessons were learned, but in the end we chose to implement our application for Android differently.

Firstly, the Windows Mobile application had different user interface guidelines. It showed data in a simple table. Our desire was to create a simple graphical user interface, heavy on large graphics, and with a more visual presentation of data. Our reasoning was that this application is meant for people with congestive heart failure, a demographic heavily populated by the elderly and other people with cognitive disabilities[13], which may not be the most technically savvy users.

On the architecture side, the Windows Mobile application was developed using a model-driven methodology. Ours is hand crafted from the bottom up, as

Android does not have strong support for model-driven development. This has a few implications for the project. It will be slower to develop, but the code will probably be more maintainable for developers taking over the code-base. Seeing as extensibility was a desire of the customer, we feel this was a fair trade-off.

3.4 Technology for ESUMSDroid

This section details the technologies used to make ESUMSDroid possible. It incorporates the development platform for our project, the integrated development environment we used and the communication protocols needed for our project to be compatible with the existing ESUMS components.

3.4.1 Development platform

During the customers planning of the ESUMS, they analyzed several different mobile platforms (details below). The goal was to find out which platform was most suitable to their needs. At the time of their analysis, Windows Mobile was deemed most suitable, and was therefore used when they created their prototype application. The main reason Android was not chosen was the lack of Bluetooth support on most devices. However, since then several new versions of Android have been released. Most specifically version 2.0 and newer. This iteration of Android brought with it the requirement of Bluetooth on all devices, making it possible to implement an Android prototype of their handheld client.

Chosen platform: Android/Java

Android is an open-source platform for smart-phones developed by the Open Handset Alliance of which Google is a prominent member[18]. Android is currently being maintained by the Android Open Source Project headed by Google[19]. At the time of writing Android has a 43.6%[20] market share in the smart phone market and a large community writing applications for the operating system.

Applications developed for the Android operating system is written in a subset of Java[4] and runs on the Dalvik virtual machine. To develop applications one uses the Android SDK, which supplies the user with many useful tools (more on this later) for development, and the ADT plug-in for Eclipse. Android also supports the use of other programming languages to a certain degree, but for this project this is not relevant.

[4]For the purposes of this project, the restrictions of the subset is not paid particular attention to because they wont affect the project

A typical Android application will consist some key components, which will be discussed below.

Activities

An Activity[21] is a component with a visual interface that will preform a specific function. An application usually has several activities, one for each function the user would want to undertake. Activities can launch each other, this way the user can navigate through the application. An activity can take several visual forms, popup boxes, alert dialogues, configuration menus and other screens.

The components that make up the visual content of an activity are called views, but are more commonly referred to as widgets. A View[22] can be a button, check box, text field and many other things, you can even define your own views. Views are organised in hierarchies, very much like swing elements from plain Java, and the developer controls the layout of the views used in an activity by XML.

One of the most important things to note about activities is the non-deterministic lifecycle[23] of an activity. Once an activity leaves the foreground of the handheld, the operating system might destroy it at any point in time should the resources taken up by the activity be needed for other things. This could cause some issues with the availability of the application, but luckily there are ways to work around this, which will be explained further in the next paragraph.

Services

A Service[24] is a component, which unlike activities, does not have a visual user interface. Instead a service will typically run in the background and preform tasks that does not require user interaction, such as calculations, playing music or data transfer. A service also gives the developer options for more direct control over its lifecycle[25], which complements the activities uncertainties very nicely. Using a service as the backbone for our ESUMSDroid will help to ensure maximum uptime for the system.

A service can communicate with an activity in several ways, these are

- Binders
- AIDL
- Messengers

Binders are a lightweight class for communication between service and activity. AIDL, or Android Interface Definition Language, is intended for cross-process communication, if you would like your service to be available for other applications. Messengers, a simpler version of AIDL used when your service only communicates with activities within its own process. One of the key issues with service-activity communication is state management, because ideally a service should preform its

function without regard for the current state of the activity. This issue is solved by using one of the three methods mentioned above.

Database

When choosing Android as your development platform you get a tightly integrated SQLite database which is easy to use from a developers point of view. SQLite is a lightweight, ACID-compliant database management system, and its small core makes it ideal for handheld devices. There are other options for databases when developing for Android, but these require a lot of extra work while achieving very little extra functionality. One thing to note is that an SQLite database created will only be accessible to the application that created it.

This covers the key components[5] in the Android framework which are of particular importance to this project.

When developing for the Android platform you will likely want to test your application throughout the development process, for this purpose an Android emulator is included in the Android SDK. However, this emulator is not without its drawbacks. It runs less than smoothly on slow computers and works less than satisfactory when working with network and bluetooth communication. Especially the problems with communication is a major drawback as that is the primary functions of our application. Luckily the debugger works nicely with just about any device that runs Android and has the capabilities to connect to a computer via USB.

Because of the requirement that our application supports bluetooth communication we need to use Android 2.0 as the minimum version requirement for our application. While this means the exclusion of any handheld device that runs a version of Android lower than 2.0, we can see in figure 3.7 that this only concerns less than 25% of the devices on the market[26], and this fraction will only decline as more and more phones with Android 2.x are released.

Other suitable platforms

When the project was presented to us, the customer had already done an analysis of most major mobile platforms in the search of the most suitable one. In addition to Android, these were iPhone, Symbian, Windows Mobile and Blackberry. Therefore these were the realistic alternatives. Also, a Windows Mobile version had already been implemented, meaning the customer was looking to support a wider range of platforms. As a platform for a proof of concept prototype however, Android seemed to be the best choice at the time, due to its availability and functionality.

[5] There are plenty more key components in the Android framework, but these are not relevant to our project. See `http://developer.android.com/guide/topics/fundamentals.html` for additional details.

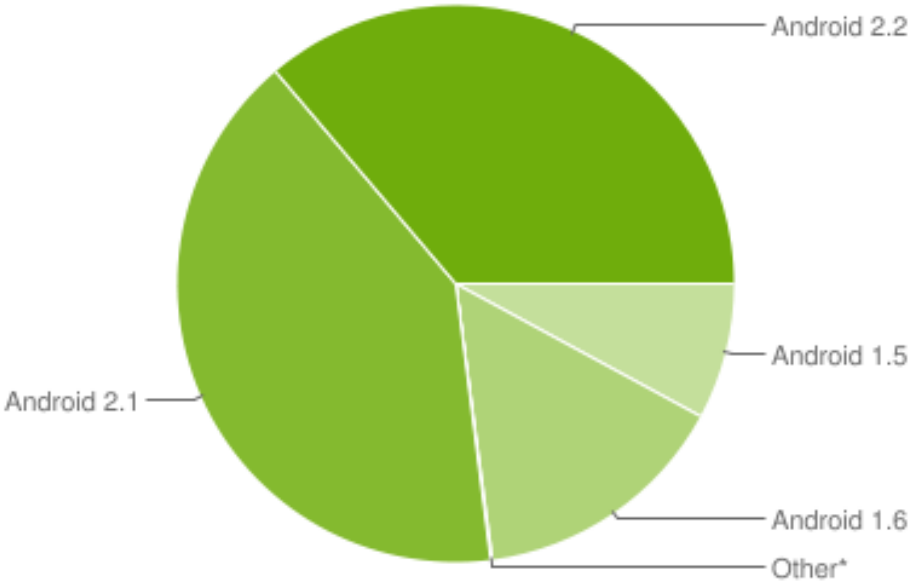

Figure 3.7: An overview of the current distribution of Android versions[3].

A table summarizing the customers analysis can be found in the appendices of [1] also repeated in table 3.1 [6].

3.4.2 Integrated development environment

When developing computer programs, especially programs of a certain size and complexity, the help of an IDE is invaluable. An IDE usually comes with code highlighting, interpreter and/or compilers, package explorers, debugging tools and more. These are tools that help immensely with the development process, so choosing a well suited IDE is integral to the success of the project.

Chosen IDE: Eclipse

The chosen IDE, Eclipse, is the de facto IDE for java development and also the IDE 1st year computer science students learn to use at NTNU, which means that most of the team was already familiar with it. Eclipse is also widely used, which means that troubleshooting and guides are readily available and easily accessible.

Still, one of the main reasons for choosing this particular IDE is that it is the recommended IDE when developing Android applications[27]. An Android Development Tools (ADT)[28] plug-in already exists for Eclipse, developed and

[6] One thing to note is that at the time Android had not released their 2.0 update, so the bluetooth API was not available.

Table 3.1: The mobile platform study preformed by the customer. Found in [1]

OS	Open for app. develop.	"Native" languages	IDE	Multi-tasking	GPRS + GSM simult.	Java MIDP	Device vendors	BT progr.	Web service client support	3. party lib licenses
iPhone	Dev. license required	Obj. C, C	XCode on Mac	Limited multi-tasking, full multi-threading	Limited	No	Apple	Only for selected accessories?	3. party tools (e.g. wsdl2objc)	OK
Symbian	Yes	C, C++, (Java)	Eclipse	Multi-tasking and Multi-threading	Yes, see note below (*)	Yes	Nokia, Ericsson, Samsung, LG	JSR82 for Java. SDK for C++	Nokia WSDL tool for C++	
Windows Mobile	Yes	C#, Visual Basic	Visual Studio on Windows	Multi-tasking and multi-threading	Yes	Not included with all devices. Available from 3.rd parties	Samsung, LG, HTC, Motorola, Sony Ericsson, HP, :.	Yes	Yes	?
Android	Yes	Android Java	Eclipse or other Java IDE	Multi-tasking and multi-threading	Yes	?	HTC, (Samsung, LG, Sony Ericsson, Motorola)	No-not in current version	kSOAP (3. party)	OK
Black Berry**	Yes	Java	JDS / Eclipse	Multi-tasking and multi-threading	Yes, see note below (*)	CLDC 1.1, MIDP 2.0	BlackBerry	Serial port programming for Bluetooth in some models.	JSR172, WTK	RIM if using the enterprise solution.

*To be able to talk and use data traffic simultaneously using a CDMA network, the CDMA EV-DV version is required. Other CDMA standards do not support this. UMTS and GSM/GPRS do support talk and data traffic simultaneously. However, it is assumed that most networks in North America run on CDMA technology.

**Blackberry phones are not yet readily available in Norway.

*** Java MIDP is highly portable to different phones. Requirement is that phone runs a Java Virtual Machine.

maintained by Google. The use of Eclipse's ADT plug-in helps immensely with the development process, especially the inclusion of an Android emulator, .apk-exporting, LogCat (a debugging tool included in the ADT plug-in) and more.

Eclipse is also multi-platform, and runs wherever a JVM has been installed. This meant that everyone on the team could work in the same environment and share experiences and help each other more easily. Also, to ensure that we follow a certain coding standard, Checkstyle[29] is used. It is a development tool that puts certain customizable rules on how the source code should be written. Mostly it relates to variable and method naming conventions. It comes in the form of an Eclipse plugin. It helps us write similar code and makes it easier to read each

others code. As we are handing over our source code to the customer at the end of project, it is also helpful if they intend to review and build on our code.

Other IDEs

Because the ADT plug-in only exists for Eclipse, any other IDE would require significant overhead in the development process. Combined with the previous experience with Eclipse, there was never any real competition to Eclipse as the groups IDE.

3.4.3 Communication with sensors

The purpose of the application developed during this project is to collect data from an external sensor and transmit this data to a remote server. The communication between the external sensor and the application ideally needs to follow some sort of wireless communication protocol. Currently, the most common short-range protocol is arguably Bluetooth. Along with the fact that more or less every handheld device with Android supports bluetooth, and the two already existing external sensors we are getting to use during this project(see section 3.3.2 for details) are based on bluetooth, makes it an obvious choice.

Chosen communication protocol: Bluetooth

Bluetooth is a technology for short-range wireless communication, and was selected for the ESUMS Chest Unit. Thus, our application will have a component for communicating with the chest unit (and other devices) using this technology. Bluetooth is a complex technology, attempting to unify short-range communication under a single umbrella name. This wound up having some implications for our project.

First, Bluetooth supports several communication profiles. One of these, called HDP or Health Device Profile, is specifically tailored for medical devices, as implied by the name. The ESUMS project hopes to make use of this standard, but due to its relative youth it is not widely supported by devices outside the medical field. Android for instance, does not support it. In the end, we wound up with the SPP (Serial Port Profile), which emulates the RS-232[30] serial port. This seemed sufficient for the needs of our application.

Further, Bluetooth obviously puts some restrictions on range, being a wireless technology. It has several defined classes, each with different supported ranges. At the beginning of the project, and throughout the entire preliminary study, it was unknown which class the Chest Unit fell into.

Other communication protocols

The most relevant alternatives to bluetooth include IrDA and Wi-Fi. However both of these have some limitations making them unsuitable compared to bluetooth. IrDAs main problem is the need for a direct line of sight between communicating devices. This is very inconvenient in a system designed to be used several hours each day. Maintaining line of sight would be time consuming and would result in loss of data if line of sight is disrupted. Not to mention the limited amount of Android devices supporting IrDA.

Wi-Fi on the other hand is supported by more or less every Android device, just like Bluetooth. It also does not require direct line of sight and has superior range. However, compared to Bluetooth, it consumes more battery and is initially less secure. Wi-Fi connections can be hijacked, unless significant time is spent implementing secure connections. Bluetooth has some security built in and can easily be configured to only communicate with pre-determined devices.

3.4.4 Server communication

Communication with the ESUMS server will have to be done over the internet using SOAP to communicate with the services offered. Internet communication is not an issue with most handheld devices as every device running Android has hardware that supports 3G or GPRS communication. Officially Android does not support SOAP, which means that no official Android SOAP library currently exists. However, an implementation of kSOAP2[17] has been tailored towards Android, and therefore makes an obvious choice for our project. kSOAP is a SOAP library designed specifically for mobile Java environments. For our implementation we therefore need to make the application compatible with this mapping. SINTEF's implementation of WSDL is used together with SOAP.

3.5 Standards

This chapter includes more information on the different standards used in the ESUMS and in ESUMSDroid.

3.5.1 Standard for communication between medical devices: IEEE 11073

The IEEE publishes several standards. One of these, number 11073, pertains to communication between medical devices, their data-exchange format and gives guidelines on architecture.

IEEE 11073, billed as a complete solution to medical device communication, is a weighty tome. Most parts of it did not affect our application, being completely unrelated to our problem domain. Some parts were more relevant, detailing data representation and communication requirements. In the end we decided the standard was simply to complex to be followed to the letter, for a project as small as ours. We did however try our best to follow the recommendations put forth, and as such studying this standard was not a complete waste of time[7].

3.5.2 HL7

HL-7 is a standards organization which works to produce standards applicable to the field of health-care. Our project does not make use of these standards directly, however FreeMpower, the web service framework on which the ESUMS server is based, does. Thus our communication protocol with the web service makes use of HL-7 published standards.

3.6 Preliminary study conclusions

We chose to use scrum as our development methodology for this project to make the impact of uncertainties within the project domain as small as possible and to facilitate a continuous communication with the customer throughout the development process. To help us in our development process we chose a set of collaboration and communication tools. The choice of these tools were based on past experiences with project work from the members of our group. We did not have much choice in the decision of development platform because the customer requested an application for the Android platform. This decision also meant that we were bound to writing mainly in Java and using Eclipse as the IDE for this. To be able to function in cooperation with the existing parts of ESUMS we were also required to use bluetooth for device communication and SOAP for communication with the ESUMS server.

[7] At a point during sprint 1, we were informed by the customer that we would not be following this standard, however not before having studied it for some time.

Chapter 4

Requirements Specification

This chapter aims to describe the requirements set upon the system. The requirements specification details how the finished system should function and what is to be expected from it. This is done through usage scenarios, use cases and functional and non-functional requirements. At the top we have usage scenarios, which are a description of how the system should function from the end user's point of view. Use cases go into more detail about how each part should function and what should and should not happen during different flows of events. At the bottom we have the functional requirements, which are very detailed descriptions of how each part of the system should function. These requirements together give the final system as a whole. Based on this we have also created a product backlog, to have short and precise tasks for implementation. This backlog is used during the sprints. It consists of a list of items which are to be implemented in the system, where each item describing only a small, specific part of the finished system. This makes it easier to implement in steps, and making sure each part is properly handled.

This chapter will consist of these sections:

- **4.1 Functional requirements:**
  The functional requirements section includes a section for usage scenarios, one for use cases, and one for the functional requirements. Based on this we have also created a detailed product backlog. These are the items directly used during implementation.

- **4.2 Non-functional requirements:**
  The non-functional requirements section will cover requirements related to the quality of the product.

4.1 Functional requirements specification

Functional requirements The point of functional requirements is to describe the behaviour of the envisioned finished system. This is done by splitting up the system into smaller parts, and detailing how these parts should respond to input, what they should output and so on. Ideally they should describe the system at fairly low level. However, getting to a final list of functional requirements is a long process. To get a full understanding of a system, one needs to have a starting point. The detail needed for functional requirements is often not available early in the process, and so one starts with real-life scenarios of what the system should be able to do. That is why we first created a set of usage scenarios. We needed these to make sure we knew and understood what sort of problems the system was meant to handle. From here, we could create a set use cases. These take sections of usage scenarios and lay them out in more detail, with precise flows of events. After completing these two documents, one has a thorough understanding of the system. Therefore making the creation of an as complete list of functional requirements as easy as possible.

Relationships between usage scenarios, use cases and functional requirements Figure 4.1, along with tables 4.1 and 4.2, show the relationships between the usage scenarios, use cases and functional requirements. They are meant to give a quick overview of how each requirement covers the necessary functionality of the finished system. Our whole approach while creating and selecting scenarios and requirements has been based upon figure 4.1.

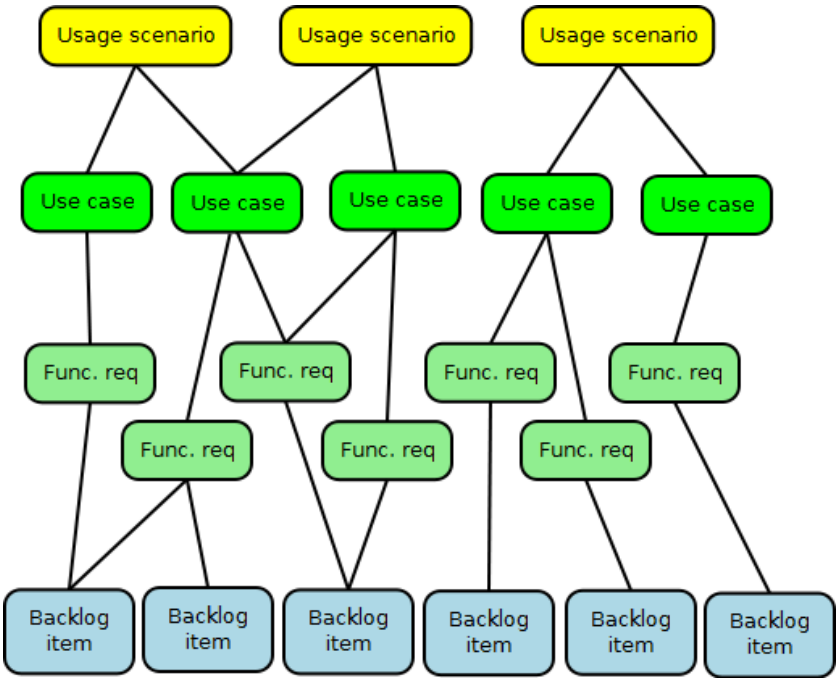

Figure 4.1: The correlation between usage scenarios, use cases, functional requirements and backlog items. Based on [4]

Table 4.1: Relationship between usage scenarios and use cases

Usage Scenario	Usage Scenario section	Covered by use Case #
A1	A1.3	1
	A1.4	3, 11, 12
	A1.5	3, 5
	A1.6	2, 4
P1	P1.6	1, 3
	P1.7	10
	P1.8	2, 4
1	1.1	1
	1.3	2, 4
2	2.1	1
	2.3	3, 6, 8
3	3.1	1
	3.4	13
	3.5	13
4	4.3	1, 3, 6, 7
5	5.2	1
	5.3	3, 5
	5.4	2, 4
6	6.3	9
	6.5	1

Table 4.2: Relationship between use cases and functional requirements

Use Case	Functional requirement
1	REQ073 REQ122
2	REQ002 REQ003 REQ026 REQ027 REQ077
3	REQ024 REQ073 REQ122
4	REQ007 REQ026 REQ027
5	REQ021 REQ024
6	REQ020 REQ079
7	REQ079 REQ121
8	REQ079 REQ121
9	
10	REQ021 REQ024 REQ122
11	
12	
13	REQ027

4.1.1 Usage Scenarios

Usage scenarios describe the system functionality at a high level. Their purpose is to give examples of the interaction between users and the system in a realistic setting. They try to describe actual uses of the finished system. Usage scenarios are created early in requirements specification phase. A complete overview of the system is still lacking during this phase, and that is what the usage scenarios aim to give. It gives the developers an idea of what should be accomplished.

The usage scenarios start with a user of the system. This user has a problem, which the system should be able to solve. Based on this prerequisite, the scenarios go through the steps needed to reach a solution. They do not go into detail, and do not describe alternative flows of events. They are a strict description of what the system should be from a users perspective.

Customer Created Usage Scenarios The following usage scenarios were created by the customer and are described in [1]. They are two examples of how the customer imagined the system should behave during their own planning and requirements phases.

Passive usage scenario

1. It is in the morning. Matthew (74 year old with congestive heart failure) turns the chest unit on by pressing and holding the on/off button. He then fastens it securely around the chest.
2. Matthew ensures that the battery is fully charged by observing that a light on the belt is turned on.
3. He is not really interested in viewing the data the chest unit is measuring. Thus, he does not turn the handheld on. This is ok as the handheld can log the measured data for a full day if necessary.
4. He wears the belt for the whole day according to the agreement with his nurse.
5. He then takes the belt off.
6. He starts the ESUMS handheld application. After a short period he can see from the screen that the chest unit has started transmitting data to the handheld.
7. He is instructed to weigh himself every day, so he does this (whilst transmission is still going on). He turns on the electronic weight scale, presses the "other sensors" button on the handheld's screen, and follows the instructions.

8. He waits for a while to ensure that all the measurements have been transmitted to the server, before he turns off the chest unit and the handheld.

Active usage scenario

1. It is in the morning. Jim (63 years with congestive heart failure) fastens the chest unit securely around his chest. He then turns the chest unit on by pressing the on/off button.

2. Jim ensures that the battery is fully charged by observing that the light on the belt is turned on.

3. He likes to follow up on his heart rate and activity during the day, via the handheld screen. He starts the ESUMS application on the handheld.

4. He ticks off the LIVE DATA box on the handheld, as he wants to follow up on the data while he is out walking. He knows that it can take a short period of time before he actually sees the data on the screen, since the chest unit needs to "wake up" before it can start transmitting data continuously. He knows that if he does not want to wait, he can force the chest unit to send live data right away by pressing the on/off button on the belt twice and rapidly1.

5. He then starts off on his walk. Coming to the foot of a slope he checks on his heart rate. At the top of the hill, panting from the exercise, he checks again to see his max pulse.

6. He knows that the data is sent to server at regular intervals, so he does not have to worry about that.

Additional Usage Scenarios The following usage scenarios were created by the development team to help gain a deeper understanding of the functionality of the handheld application. Initially, the usage scenarios from the previous section were the only ones we had. Deliverable 1 [4] was not given to the group until at a later stage, after the creation of the following scenarios. Therefore we decided to continue to base our work on these new scenarios. They follow the same structure as the customer-created scenarios, although with a slightly more direct approach, in that they each cover a specific, isolated task and do not go further. The main system functionality derived from the usage scenarios is the same. Where the customer-created scenarios and our scenarios differ is mostly with regards to additional features, such as questionnaires and a messaging system. However, after

a discussion with the customer, it became clear they did not want these features to be implemented during this project, as they did not have the infrastructure in place for them.

Usage Scenario 1: Sending data to the ESUMS server

1. Pat turns the chest unit on, and starts the application on the handheld
2. Pat then leaves the handheld on the desk while he works for a couple of hours
3. At regular intervals during the past hours, the application automatically sends all the recorded data to ESUMS server

Usage Scenario 2: The battery on the chest unit gets low

1. Kim turns the chest unit on, and starts the application
2. Kim then leaves the handheld in her purse
3. The chest unit gets low on battery
4. The handheld alerts Kim to the low battery by making a sound and vibrating

Usage Scenario 3: The handheld leaves transmission range

1. Joe turns the chest unit on, and starts the application
2. Joe then leaves the handheld on the desk while he works
3. Joe gets up to go to the toilet in the floor above his office
4. The chest unit leaves the transmission range of the handheld
5. The handheld alerts Joe with sound and vibration that the chest unit is no longer within range

Usage Scenario 4: Patient with poor eye sight

1. Jim puts on the chest unit.
2. He is not sure the battery is fully charged because his eye sight is poor.
3. He turns on the ESUMS handheld application to get an update on the battery status of the belt unit, and make sure the handheld is charged so he can transmit collected data later in the evening. The handheld shows both batteries are fully charged.
4. Satisfied with the battery level, he continues with his day.

Usage Scenario 5: Checking normally recorded data regularly

1. John turns on the chest unit. John likes to check the status of his measurements at regular intervals during the day, but he's not concerned with the data being current. He only cares that the measurements are as they should be over time, and he likes to control that everything is working properly

2. In the afternoon, he turns on the handheld application, which automatically connects to his chest unit

3. Data from the last few hours is transmitted to the handheld, and John views the data in an organized view on the handheld. He likes what he sees, and turns the handheld off

4. Later in the evening he repeats the process. Still satisfied with the results, he turns off the handheld and his chest unit before going to sleep

5. The handheld transmitted his data to the server automatically when he had it on during the day, so he does not worry about checking that

Usage scenario 6: Application configuration scenario

1. Administrator has a patient. This patient needs to be monitored using the ESUMS system.

2. Administrator gets a chest unit and handheld device.

3. He then enters the patients credentials into the handheld configuration, including patient's real name, user name and password.

4. He also enters in the unique n-digit letter and number string belonging to the selected chest unit. With the configuration complete, the handheld device and the chest unit are paired together and only work with each other.

5. He checks the devices are correctly connecting to each other.

6. The patient is then instructed in how to use the system by the administrator.

4.1.2 Use cases

Use cases give more detail than the usage scenarios. Their purpose is to describe select parts of the usage scenarios at a more detailed level. They initially follow a set path, but with alternative flows and descriptions of what should and should not happen during a given flow of events, including some error handling. They

are all based on a usage scenario, and can be directly linked back to a specific scenario from the previous section. A table showing the relationships between usage scenarios and use cases can be found in the beginning of this chapter.

We created initial drafts of the use cases early on, based on our understanding of the project at the time. When we got more documentation from the customer, we compared requirements with those found in [4], and felt there were no specific conflicts between them. The main functionality was detailed correctly, and where our use cases and the customers use cases differed, was mainly with regards to additional functionality decided not relevant to our project. Therefore we chose to stick with the use cases we had the clearest knowledge of.

Figure 4.2 is an overview of the application from a patients point of view. Figure 4.3 and tables 4.3 - 4.15 go more into detailing the flow of events.

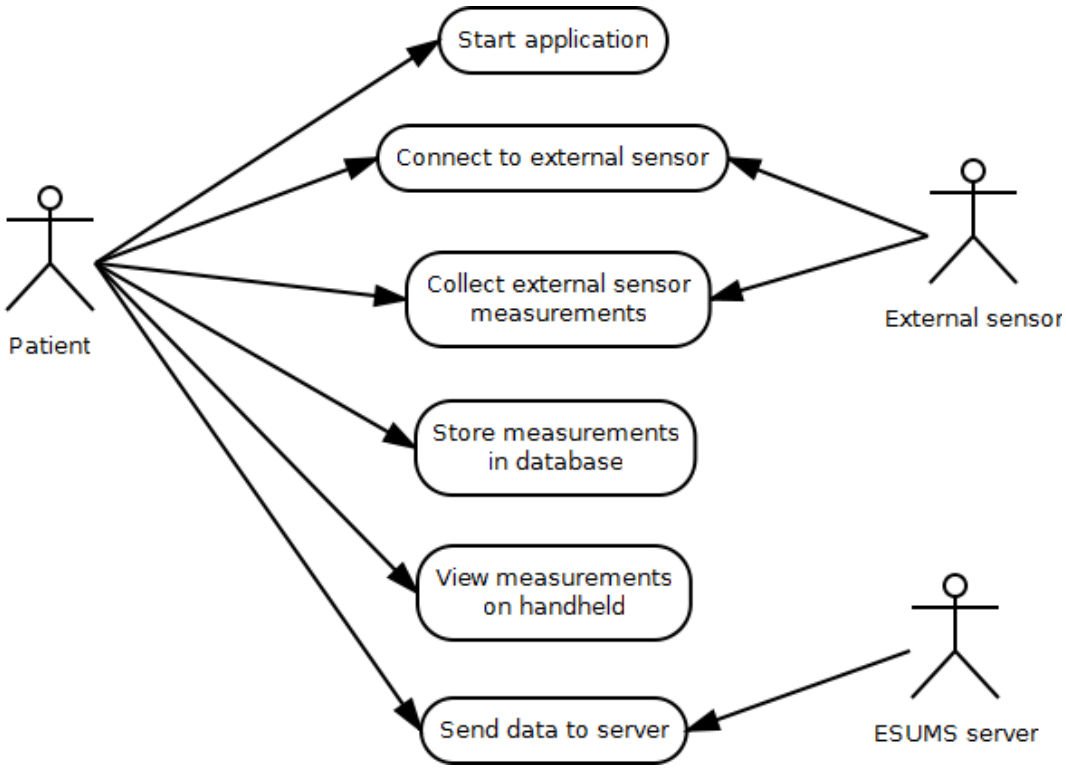

Figure 4.2: Application overview from patient point-of-view

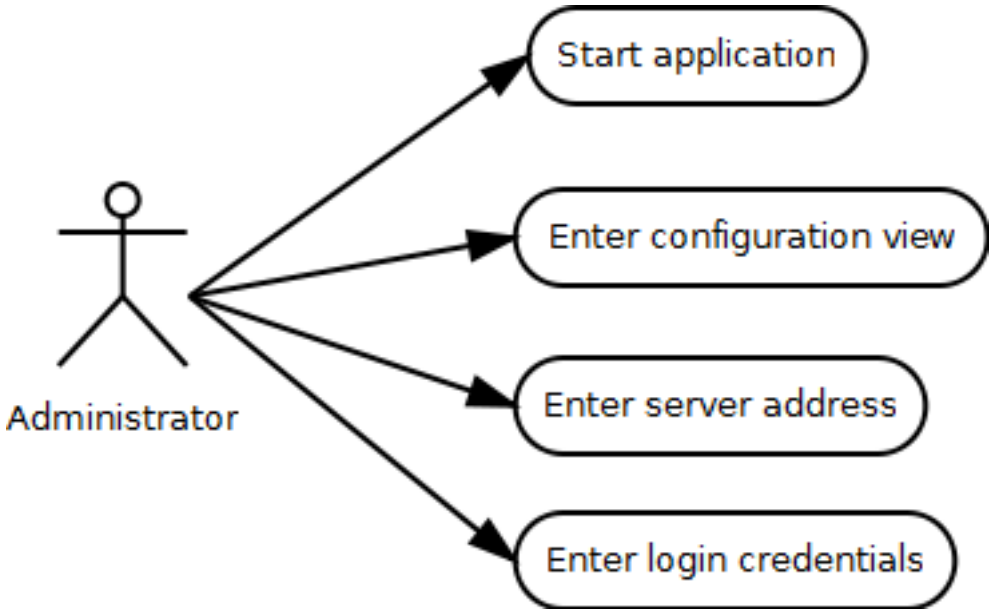

Figure 4.3: Application usage from administrator point-of-view

Table 4.3: Connection between Handheld and Bluetooth device

Use case ID	1
Use case goal	Connection between Handheld and Bluetooth device
Participants	Handheld application, Bluetooth device
Preconditions	Handheld is turned on with Bluetooth enabled, Application is running, Bluetooth unit is turned on, Handheld and Bluetooth device have been paired in the past.
Flow of events	1. Application prompts the Handheld to search for paired devices. 2. Handheld automatically searches for paired devices within its range. 3. Handheld connects to the device.
Extensions	2a. No nearby chest units are found. Repeat step 2 for a set number of times. 2b. No devices found after a set number of times, notify user. 3a. Connection fails. Repeat step 3 for a set number of times. 3b. After a set number of repeats, and still no connection, notify user.

Table 4.4: Connection between Handheld and ESUMS server

Use case ID	2
Use case goal	Connection between Handheld and ESUMS server.
Participants	Handheld application, ESUMS server
Preconditions	Handheld is turned on with data transfer enabled, Application is running, ESUMS server credentials have been configured in the Application.
Flow of events	1. Application requests a internet connection from the Handheld. 2. Application gets a connection to the ESUMS server. 3. Application logs in with credentials from configuration file.
Extensions	2a. If no connection is made, repeat step 2 a set number of times. 2b. If, after a set number of repeats, no connection is made, notify user. 3a. If authorization fails, notify user.

Table 4.5: Successfully received data from Bluetooth device

Use case ID	3
Use case goal	Successfully received data from Bluetooth device.
Participants	Handheld application, Bluetooth device
Preconditions	Use case 1
Flow of events	1. Application requests data from the Bluetooth device. 2. Data is timestamped and stored.
Extensions	1a. If request fails, retry a set number of times. 1b. If, after a set number of retries, the request still fails, notify user.

Table 4.6: Successfully send data to ESUMS server

Use case ID	4
Use case goal	Successfully send data to ESUMS server.
Participants	Handheld application, ESUMS server
Preconditions	Use case 2
Flow of events	1. Application retrieves unsent data from storage. 2. Application sends data to ESUMS server. 3. Application deletes successfully sent data from storage.
Extensions	2a. If data is not successfully sent, retry a set number of times. 2b. If data is not successfully sent, after a set number of retries, store for later retry and notify user.

Table 4.7: Display recent data on Handheld

Use case ID	5
Use case goal	Display recent data on Handheld
Participants	Handheld application
Preconditions	Use case 3
Flow of events	1. Application requests new data from storage. 2. Application displays data on Handheld screen.
Extensions	1a. No data new data is found, notify user.

Table 4.8: Successfully reading the battery status of the Chest unit

Use case ID	6
Use case goal	Successfully reading the battery status of the Chest unit.
Participants	Handheld application
Preconditions	Use case 3
Flow of events	1. Application reads the current battery status of the Chest unit from recently received data. 2. Application stores battery status. 3. Application displays battery status on Handheld screen.
Extensions	None

Table 4.9: User is informed of the battery status in a non-visual way

Use case ID	7
Use case goal	User is informed of the battery status in a non-visual way.
Participants	User, Handheld application
Preconditions	Use case 6
Flow of events	1. User clicks the battery status indicator on Handheld screen. 2. Application plays a sound depending on battery status.
Extensions	None

Table 4.10: User is informed of the battery status when Application is in inactive mode

Use case ID	8
Use case goal	User is informed of the battery status when Application is in inactive mode.
Participants	User, Handheld application
Preconditions	Use case 6 and Application is in inactive mode.
Flow of events	1. Application detects low battery on Chest unit 2. Application notifies the User.
Extensions	None

Table 4.11: Successfully modify the configuration file

Use case ID	9
Use case goal	Modify the configuration file.
Participants	Administrator, Handheld application
Preconditions	Handheld is turned on, Application is running
Flow of events	1. Administrator selects configure application on the Application. 2. Application prompts Administrator for credentials. 3. Administrator modifies the configuration file.
Extensions	2a. Authorization fails, repeat step 2. 3a. Configuration file does not exist, start wizard for first-time configuration.

Table 4.12: View data from other sensor

Use case ID	10
Use case goal	View data from other sensor.
Participants	Handheld application, User
Preconditions	Use case 1
Flow of events	1. User clicks on the other sensors button. 2. Application displays other sensors on Handheld screen. 3. User selects desired other sensor. 4. Application displays recent other sensor data on Handheld screen.
Extensions	2a. No external sensors have been configured, notify user. 4a. No recent data exists for desired external sensor, notify user.

Table 4.13: Switch between Live Mode and Normal Mode

Use case ID	11
Use case goal	Switch between Live Mode and Normal Mode
Participants	Handheld application, User
Preconditions	Handheld is turned on, Application is running, Live Mode is selected.
Flow of events	1. User deselects Live Mode. 2. Application enters Normal Mode
Extensions	None.

Table 4.14: Switch between Normal Mode and Live Mode

Use case ID	12
Use case goal	Switch between Normal Mode and Live Mode
Participants	Handheld application, User
Preconditions	Handheld is turned on, Application is running, Normal Mode is selected.
Flow of events	1. User selects Live Mode. 2. Application enters Live Mode
Extensions	None.

Table 4.15: Notify user if connection between handheld and bluetooth device is lost

Use case ID	13
Use case goal	Notify user if connection between handheld and bluetooth device is lost
Participants	Handheld application, Chest unit/Other sensor
Preconditions	Use case 1
Flow of events	1. Bluetooth device goes out of range of handheld application 2. Handheld plays a sound and vibrates
Extensions	None.

4.1.3 Functional requirements

This section details the functional requirements for our application. The requirements were created and detailed by the customer in ESUMS Deliverable 1 System Requirements v1.0 [4]. We took these requirements and, in discussion with the customer, we decided what was and what was not to be implemented. This section lists all the relevant requirements, along with comments on implementation details. Mainly these comments consist of explanations on whether or not the requirement was to be implemented. If the requirement was not being implemented, we added a small note explaining why. Each requirement has a status and a priority. These values were set on the requirements by the customer.

Chest unit functional requirements

Requirements covering functionality of the chest unit from the handhelds point of view.

- **REQ020 - Chest unit operation time 24 hrs**
  Status: Proposed
  Priority: High
  Version: 1.0
  Detail: The chest unit must operate for 24 hours on fully charged batteries in normal measurement mode. In normal measurement mode data is stored locally on the chest unit and transmission to the hand-held is done in batches, as opposed to continuous mode where data is transmitted real-time to the hand-held.
  Comment: Only continuous (live) mode will be implemented in the handheld, as the data from the chest unit lacks time stamps.

Handheld functional requirements

Requirements covering the functionality of the handheld application

- **REQ024 - Handheld vital signs application**
  Status: Proposed
  Priority: High
  Version: 1.0
  Detail: The handheld must run an application that shows the measured sensor data. Views for both chest unit data and external sensor data must be included.
  Comment: Will be implemented

- **REQ069 - Handheld questionnaire application**
  Status: Proposed
  Priority: High
  Version: 1.0
  Detail: The handheld must run an application for filling out and managing questionnaires.
  Comment: Will not be implemented. Customer does not want it at this point.

- **REQ122 - Attachment of external sensor**
  Status: Proposed
  Priority: High
  Version: 1.0
  Detail: It must be possible to connect an externalCOTS sensor device to the handheld with wired or wireless connection.
  Acceptance criteria: One will be selected for integration and demonstration in the prototype.
  Comment: Support for the NONIN sensor will be implemented. No other external sensors (not counting the chest unit) will be implemented at this time, but it will be relatively simple to implement support for new ones later.

- **REQ121 - Handheld system management application**
  Status: Proposed
  Priority: High
  Version: 1.0
  Detail: The handheld must provide functions for the user to view ESUMS system status regarding connection to and status of communication with chest unit and external sensor devices.
  Comment: Will be implemented

- **REQ077 - Handheld handling of failing communication with server**
  Status: Proposed
  Priority: High
  Version: 1.0
  Detail: The handheld communication protocol with the server must be able to handle failing and disruptive communication.
  Comment: Will be implemented

- **REQ079 - User alerts on system status**
  Status: Proposed
  Priority: High
  Version: 1.0

Detail: The user must be alerted by audio and/or visual signaling on handheld about errors or changes in communication and system status, including battery status.
Comment: Will be implemented. Note that chest unit currently does not transmit proper battery level notifications. This means monitoring of battery status is not possible at this time.

- **REQ021 - Continuous measurement view**
  Status: Proposed
  Priority: Medium
  Version: 1.0
  Detail: It should be possible for the user to view the sensor-data real-time, when the data is continuously transmitted from sensor-source to handheld.
  Comment: Will be implemented

- **REQ082 - User reminders**
  Status: Proposed
  Priority: Medium
  Version: 1.0
  Detail: The handheld healthcare self-management applications should run reminders to alert the user to carry out planned healthcare activities.
  Comment: Will not be implemented; not needed according to customer.

- **REQ086 - Deviation analysis application**
  Status: Proposed
  Priority: Medium
  Version: 1.0
  Detail: Received sensor measurements should be analysed real-time on the handheld and compared to the patient's "normal" value range (i.e the threshold values)
  Comment: Thresholds for normal range for various measurement data will probably be implemented, but there will be no functionality for learning the normal range. (Also handled on nurse's application.)

- **REQ025 - Handheld exercise application**
  Status: Proposed
  Priority: Low
  Version: 1.0
  Detail: The handheld may run an application that provides the user with information about exercise plans and status.
  Comment: Will not be implemented. Customer not interested.

- **REQ085 - Handheld health incident application**
  Status: Proposed
  Priority: Low
  Version: 1.0
  Detail: The handheld may provide functions that allow the user to register the time of an incident (e.g. events of pain) and to let the user register a description of the incident.
  Comment: Will not be implemented. Customer does not want it at this point.

- **REQ068 - Trigger notification**
  Status: Proposed
  Priority: Low
  Version: 1.0
  Detail: The handheld may continuously analyze the received chest unit sensor data, and if the data indicates need for medical attention a notification may be triggered and sent to identified recipients, e.g. via sms.
  Comment: Will not be implemented; not prioritized in prototype, and we do not possess the medical knowledge necessary to implement this analysis.

- **REQ135 - GPS location application**
  Status: Proposed
  Priority: Low
  Version: 1.0
  Detail: It may be possible to log the users location on the handheld, using handheld internal GPS functionality.
  Comment: Will not be implemented; not prioritized in prototype

Chest unit communication link

- **REQ023 - Chest unit standard communication protocol**
  Status: Proposed
  Priority: High
  Version: 1.0
  Detail: The chest unit short range communication link must be based on a standard short range communication protocol (e.g. Bluetooth). Note that the communication link must meet the defined information security requirements.
  Comment: Will be implemented

- **REQ027 - Chest unit handling of failing communication link**
  Status: Proposed

Priority: High
Version: 1.0
Detail: When communication with hand-held fails (e.g. due to out of range or handheld power shortage), data must be stored locally on unit and transmitted when the communication link to the handheld is restored.
Acceptance criteria: The chest unit must be able to locally store processed measurement data for at least 8 hours.
Comment: Not currently supported in the chest unit, as there are no time stamps. (The data is useless without it.) Therefore we will not implement support for this in the handheld (but it can be added at a later time).

- **REQ073 - Multiple chest unit-handheld communication pairs**
  Status: Proposed
  Priority: High
  Version: 1.0
  Detail: An ESUMS chest unit must only communicate with a handheldwith which it is has been configured (by system administrator) to communicate. It should be possible to operate multiple chest unit-handheld pairs within the same communication range, without communication malfunction between the individual pairs.
  Comment: Will be implemented

Server functional requirements

- **REQ026 - HL7 compliance**
  Status: Proposed
  Priority: Medium
  Version: 1.0
  Detail: ESUMS information sent to and stored at the ESUMS server must be compliant with the HL7 standard to promote future interoperability with other information systems.
  Comment: Will not be implemented; HDP is not supported by Android.

Human factors

- **REQ136 - Handheld GUI configuration**
  Status: Proposed
  Priority: Low
  Version: 1.0
  Detail: It may be possible to configure the Graphical User Interface (GUI) individually for each patient, e.g. to disable/enable applications.
  Comment: Will not be implemented; we are only doing the one application.

Safety and hazard requirements

- **REQ128 - Accompanying documents**
  Status: Proposed
  Priority: High
  Version: 1.0
  Detail: The ESUMS chest unit and handheld must be accompanied by documents containing at least the instructions for use and a technical description. The documents should be regarded as part of the equipment. The documentation must meet the standards on documentation for medical equipment.
  Comment: A user manual will be included

Information security requirements

- **REQ002 - Identification of corresponding services**
  Status: Proposed
  Priority: High
  Version: 1.0
  Detail: Services must identify and verify the identity of corresponding services before they are allowed to communicate.
  Comment: Will be implemented

- **REQ003 - Verification of users' authorisation level**
  Status: Proposed
  Priority: High
  Version: 1.0
  Detail: Services must verify the authorization level of users before access to sensitive data can be given. This applies to both server and handheld information services.
  Comment: Only the handheld aspect of this requirement applies to us. This was discussed in a previous customer meeting, and the standard PIN code for access to the handheld's functionality was deemed adequate

- **REQ007 - Integrity and confidentiality of data**
  Status: Proposed
  Priority: Medium
  Version: 1.0
  Detail: Personal sensitive data should be integrity and confidentiality protected while stored on ESUMS platforms and transmitted over open, untrusted communication lines.
  Comment: Will not be implemented, Bluetooth + ESUMS server don't support it.

4.1.4 Product backlog

The product backlog essentially contains the functional requirements for the system, but in a quite task-oriented way. These requirements are constructed as tasks that will be implemented during development. From now on, they will be referred to as backlog tasks, or simply tasks, and the whole set of backlog tasks is known as the product backlog. At the beginning of each sprint, a subset of the product backlog will be included in the current sprint backlog, which is the set of tasks to be implemented in the current sprint.

The product backlog is split into categories based on what part of the system they are concerned with, ie. the different modules of the system. (See the software architecture in chapter 5.) This grouping gives a nice overview of the system as a whole, and is convenient to ensure that all the wanted functionality has been covered by the product backlog. Within each category, the tasks are given a priority based on importance and a complexity based on how large the task is and the estimated workload needed to fully implement it. The priorities are as follows:

- High (H): These requirements cover the basic functionality of the system, and need to be implemented for the system to function as needed
- Medium (M): These requirements should be implemented for the system to meet expectations
- Low (L): These requirements are to be implemented if there is time at the end of the project. They are mostly extra features beyond the main functions of the system.

The exact wording of the requirements is important. For instance, requirement UI2 ("The UI should be able to show data from BT units.") indicates that the user interface must have the ability to display recent data from the various bluetooth units. For this requirement to be fulfilled, it is not necessary for that data to actually be available, but the user interface (UI) must have some functionality that can display the data when it eventually becomes available (through the implementation of other requirements). This implies clearly defined interfaces.

Graphical User Interface

This part of the product backlog covers the functionality of the user interface. Due to the nature of the system, user interaction should not be required unless there are errors. Ideally the user only has to start the application, and the application automatically connects, collects and forwards data. The main point of this module is therefore to let the user inspect the data being transferred from the various bluetooth devices to the server, along with the connection status between

all the necessary devices. It should also enable access for nurses and other medical personnel to edit the configuration options.

- **UI1 - Toggle Live Mode**
  The user interface should be able to select/deselect Live Mode.
  Priority: High
  Complexity: Medium
  Related use cases: 11.1 and 12.1

- **UI2 - Display data from chest unit**
  The user interface should be able to display data from chest unit or, if no data exist, notify the user.
  Priority: High
  Complexity: Medium
  Related use cases: 5.1a and 5.2

- **UI3 - Display battery and connection status'**
  The user interface should be able to display the battery status of the chest unit, the status of the connection to the ESUMS server and the status of the connection to the Chest unit.
  Priority: High
  Complexity: Medium
  Related use cases: 1.2b, 1.3b, 2.2b, 2.3a, 3.1b, 4.2b and 6.3

- **UI4 - Display recent data from other sensors**
  The user interface should be able to display recent data from other sensor or, if none exist, notify user.
  Priority: Medium
  Complexity: Medium
  Related use cases: 10.3, 10.4 and 10.4a

- **UI5 - Display a list of available sensors**
  The user interface should be able to display a list of other sensors or, if none exist, notify user.
  Priority: Medium
  Complexity: Medium
  Related use cases: 10.1, 10.2 and 10.2a

- **UI6 - Edit configuration file**
  It should be possible to change configuration file details via the user interface by entering an admin password
  Priority: Medium

Complexity: Medium
Related use cases: 9.1,9.2,9.2a,9.3,9.3a

- **UI7 - Display different status' on demand**
  It should be possible to click on the icons for the different statuses in the user interface.
  Priority: Medium
  Complexity: Medium
  Related use cases: 7.1

- **UI8 - Fullscreen view of graph**
  It should be possible to click on each graph to open a fullscreen view of that graph.
  Priority: Medium
  Complexity: Medium
  Related use cases: 5.2

- **UI9 - Dynamically adjusting graph axes**
  The range of the graphs' axes should adjust dynamically to span the data fully.
  Priority: Medium
  Complexity: Medium
  Related use cases: 5.2

- **UI10 - Normal measurement threshold of data**
  Add thresholds for normal range of the various measurement data.
  Priority: Medium
  Complexity: Medium
  Related use cases: 5.2

Bluetooth

This part of the product backlog covers the functionality of the bluetooth module. The main task of the bluetooth module is to fetch data from various devices and parse the data before passing it on to the main part of the program (the service). This module should also facilitate easy implementation of new unknown devices.

- **BT1 - Request search for paired bluetooth devices**
  The Bluetooth module should be able to receive a prompt for a search for paired Bluetooth devices in range.
  Priority: High
  Complexity: Medium
  Related use cases: 1.1

- **BT2 - Search for bluetooth devices in range**
  The Bluetooth module should be able to prompt the Handheld to search for bluetooth devices in range.
  Priority: High
  Complexity: Medium
  Related use cases: 1.2

- **BT3 - Retry search a set number of times**
  The Bluetooth module should be able to prompt retries for a set number of times if no units are found.
  Priority: High
  Complexity: Medium
  Related use cases: 1.2a

- **BT4 - Notify no devices found in range**
  The Bluetooth module should be able to tell the service that no units were found.
  Priority: High
  Complexity: Medium
  Related use cases: 1.2b

- **BT5 - Connect to bluetooth device**
  The Bluetooth module should be able to connect to a Bluetooth device.
  Priority: High
  Complexity: Medium
  Related use cases: 1.3

- **BT6 - Retry connections a set number of times**
  The Bluetooth module should retry a set number of times if no connection is made.
  Priority: High
  Complexity: Medium
  Related use cases: 1.3a

- **BT7 - Alert user no connection was made**
  The Bluetooth module should be able to tell the service that no connection was made.
  Priority: High
  Complexity: Medium
  Related use cases: 1.3b

- **BT8 - Request data from bluetooth device**
  The Bluetooth module should be able to request data from Bluetooth device.

Priority: High
Complexity: Medium
Related use cases: 3.1

- **BT9 - Retry request for data set number of times**
  The Bluetooth module should be able to retry the request for data from Bluetooth device a set number of times, and report failure to the server if no connection is made.
  Priority: High
  Complexity: Medium
  Related use cases: 3.1a and 3.1b

- **BT10 - Get battery status from chest unit**
  The Bluetooth module should be able to read battery status of Chest unit from recently received data and report it to the service.
  Priority: High
  Complexity: Medium
  Related use cases: 6.1

- **BT11 - Read or add timestamps on sensor data if available**
  The Bluetooth module should be able to read timestamps from the sensors, or automatically time-stamp them at time of receipt if no time-stamp is available.
  Priority: High
  Complexity: Medium
  Related use cases: 3.2

- **BT12 - Alert if connection with bluetooth device is lost**
  The Bluetooth module should be able to tell the service that a connection has been lost due to being out of range.
  Priority: Low
  Complexity: Medium
  Related use cases: 13.2

Database / Persistence

This part of the product backlog covers the functionality of the database and persistence of data. The main task of this module is to be a point in a redundancy chain that covers all the links in the path from the actual sensors that record data to a nurse's computer. Considering the availability requirement described in section 4.2.1, this module is a crucial piece of the final product, and needs to be properly implemented.

- **DB1 - Store data recieved from bluetooth devices in database**
  The database should be able to store data the Application receives from the Bluetooth devices.
  Priority: High
  Complexity: Medium
  Related use cases: 3.2

- **DB2 - Retrieve unsent data from database**
  The database should be able to retrieve data which has been marked as not-yet-sent (to the ESUMS server).
  Priority: High
  Complexity: Medium
  Related use cases: 4.1

- **DB3 - Store data for long periods of time**
  The database should be able to store data over longer periods of time.
  Priority: Medium
  Complexity: Medium
  Related use cases: 4.2b

- **DB4 - Remove data already sent to the web server**
  The database should be able to delete messages which have been verified as received by the web server.
  Priority: High
  Complexity: Medium
  Related use cases: 4.3

- **DB5 - Retrieve recent data from database**
  The database should be able to retrieve recent data from a Bluetooth device or, if no data exist, return that.
  Priority: High
  Complexity: Medium
  Related use cases: 5.1, 5.1a, 10.4 and 10.4a

- **DB6 - Retrieve list of all sensors from database**
  The database should be able to retrieve a list of all sensors or, if no sensors exist, return that.
  Priority: High
  Complexity: Medium
  Related use cases: 10.2 and 10.2b

Web services

This part of the product backlog covers the functionality of the web services module. The main task of the web services module is to comunicate with an ESUMS Mpower server. The service module sends datapoints to the web services module, which authenticates with the server and transmits datapoints for permament storage and review. It must also handle disruptive communication and be able to notify the service of transmission status details.

- **W1 - Recieve connection request for web server**
  The web module should be able to receive prompt to connect to the ESUMS server.
  Priority: High
  Complexity: Medium
  Related use cases: 2.1

- **W2 - Connect to ESUMS server**
  The web module should be able to connect to the ESUMS server.
  Priority: High
  Complexity: Medium
  Related use cases: 2.2

- **W3 - Retry server connection a set number of times**
  If the web module cannot connect to the server it should retry a set number of times.
  Priority: High
  Complexity: Medium
  Related use cases: 2.2a

- **W4 - Notify user if connection fails**
  If the web module cannot connect after the set number of times it should notify the user, and offer an option to retry.
  Priority: High
  Complexity: Medium
  Related use cases: 2.2b

- **W5 - Authorize with ESUMS server**
  The web module should be able to authorize with the ESUMS server.
  Priority: High
  Complexity: Medium
  Related use cases: 2.3

- **W6 - Notify user if connection repeatedly fails**
  If the web module cannot authorize with the server it should notify the user, and have him contact an administrator (username or password are incorrectly set).
  Priority: High
  Complexity: Medium
  Related use cases: 2.3a

- **W7 - Send data to ESUMS server**
  The web module should be able to send data to the web server.
  Priority: High
  Complexity: Medium
  Related use cases: 4.2

- **W8 - Verify data has been successfully sent**
  The web module should be able to receive responses which verify the data has been transmitted and notify the service of the success.
  Priority: High
  Complexity: Medium
  Related use cases: 4.3

- **W9 - Retry sending of data a set number of times**
  The web module should be able to resend data a set number of times or, if the retries fails, notify the service of the failure.
  Priority: High
  Complexity: Medium
  Related use cases: 4.2a and 4.2b

Service

This part of the product backlog covers the functionality of the Service. The Service is the backbone of the entire system running on the handheld, it ties all the other modules together and manages the flow of the system functionality.

- **S1 - Toggle between Live mode and Normal mode**
  The service should be able to switch from Live Mode to Normal Mode
  Priority: High
  Complexity: Medium
  Related use cases: 11.2

- **S2 - Toggle between Normal mode and Live mode**
  The service should be able to switch from Normal Mode to Live Mode

Priority: High
Complexity: Medium
Related use cases: 12.2

- **S3 - Tell web module to send data to ESUMS server**
  The service should be able to prompt the web module to send data to the ESUMS server.
  Priority: High
  Complexity: Medium
  Related use cases: 4.1

- **S4 - Relay connection and authorization status from web module to user interface**
  The service should be able to relay connection and authorization status notifications from the web module to the user interface.
  Priority: High
  Complexity: Medium
  Related use cases: 2.2b, 2.3a and 4.2b

- **S5 - Recieve notifications from web module and prompt database to delete sent entries**
  The service should be able to receive success notification from web module and prompt the database to delete successfully sent messages.
  Priority: High
  Complexity: Medium
  Related use cases: 4.3

- **S6 - Relay data requests from user interface to database**
  The service should be able to relay a request for Bluetooth device data from the user interface to the database.
  Priority: Medium
  Complexity: Medium
  Related use cases: 5.1, 5.1a, 10.4 and 10.4a

- **S7 - Detect chest unit battery levels**
  The service should be able to detect that the Chest unit battery is low.
  Priority: Medium
  Complexity: Medium
  Related use cases: 6.2 and 8.1

- **S8 - Notify user interface of battery status**
  The service should be able to tell the user interface of the battery status.
  Priority: Medium

Complexity: Medium
Related use cases: 6.3

- **S9 - Sound notifications**
  The service should be able to play a sound to notify the User.
  Priority: Medium
  Complexity: Medium
  Related use cases: 7.2

- **S10 - Notify user of low battery level**
  The service should be able to notify the User that the Chest unit battery is low.
  Priority: Medium
  Complexity: Medium
  Related use cases: 8.2

- **S11 - Prompt connections to ESUMS server**
  The service should be able to prompt the web module to connect to the ESUMS server.
  Priority: Medium
  Complexity: Medium
  Related use cases: 2.1

- **S12 - Relay device list requests from user interface to database**
  The service should be able to relay a request for a Bluetooth device list from the user interface to the database
  Priority: Medium
  Complexity: Medium
  Related use cases: 10.2 and 10.2a

- **S13 - Prompt connections to bluetooth devices**
  The service should be able to prompt the Bluetooth module to connect to a Bluetooth device.
  Priority: Medium
  Complexity: Medium
  Related use cases: 1.1

- **S14 - Relay connection status from bluetooth module to user interface**
  The service should be able to relay connection status notifications from the Bluetooth module to the user interface.
  Priority: Medium

Complexity: Medium
Related use cases: 1.2b, 1.3b and 3.1b

- **S15 - Relay storage requests from bluetooth module to database**
  The service should be able to relay storage requests from the Bluetooth module to the database.
  Priority: Medium
  Complexity: Medium
  Related use cases: 3.2

Overall system tasks

This part of the product backlog covers overall system functionality. The main purpose of this section is to have some concrete tasks for integration of the different modules, and not so much covering a single function of the system as opposed to the other sections of the product backlog. They are needed towards the end of the development phase, when all the other backlog items are more or less fully implemented. These tasks make sure that each part of system properly connects with all the other modules it needs to communicate with. Essentially ensuring the system is complete.

- **I1 - Service and user interface should talk to each other**
  The service and the user interface should be able to talk to each other and exchange necessary information and objects.
  Priority: High
  Complexity: High

- **I2 - Service and bluetooth module should talk to each other**
  The service and the Bluetooth module should be able to talk to each other and exchange necessary information and objects.
  Priority: High
  Complexity: High

- **I3 - Service and storage module should talk to each other**
  The service and the storage module should be able to talk to each other and exchange necessary information and objects.
  Priority: High
  Complexity: High

- **I4 - Service and web module should talk to each other**
  The service and the web module should be able to talk to each other and exchange necessary information and objects.

Priority: High
Complexity: High

4.2 Non-functional requirements specification

This section aims to describe the non-functional requirements of the system. Mainly this is done through the use of quality attribute scenarios. As opposed to the functional requirements, quality attribute scenarios do not try to describe what the system should and should not do, but rather what kind of qualities the system should possess. This is related to such factors as availability, performance and modifiability.

4.2.1 Quality attributes

The quality attribute scenarios described in this section are based on the templates and format found in the book Software Architecture in Practice by Len bass et al.[31]
We have selected three main categories for our quality attributes; modifiability, performance and availability. We feel these categories are most important for our system, and they handle what to expect from the finished product.

Modifiability Table 4.16 refers to the quality attributes modifiability. The main focus for our applications modifiability is to make it easy to add future COTS bluetooth devices. The finished product contains implementations of the chest unit and Nonin sensor fully integrated and functional into the system. However, the customer might want to supplement these sensors in the future, and therefore implementation has been done with focus on the extensability of the bluetooth module.

Performance Tables 4.17, 4.18 and 4.19 describe how the system should respond performance-wise. Mostly they cover issues regarding responsiveness while using the application. The user should be able to count on the system to successfully collect and forward all data within a reasonable amount of time. Details can be found in the tables below.

Availibility Table 4.20 shows a measurement of the availability of the system. However, this relates to the system as a whole, including the ESUMS server and chest unit/Nonin device. Within the scope of this project it is not a realistic goal to test it sufficiently. Still it is a useful estimate and should be kept in mind during the development phase. The only thing relating to this we can control fully

is minimizing data loss on the handheld application. This is done by storing all incoming data in a database, and not remove anything from the database until the application has verified that the data has been forwarded to the server.

Table 4.16: Adding new external sensors

ID	M1
Source	Developer
Stimulus	A new external sensor is created
Artifact	Application source code
Response	Connection and handling of a new external sensor with new functionality needs to be added to the handheld application
Response measure	A well documented external sensor should take no more than 6 hours to fully implement into the application

Table 4.17: Bluetooth transmission

ID	P2
Source	End user
Stimulus	User connects application to chest unit/external sensor
Artifact	Finished application
Response	Application recieves measured data and stores and displays new data
Response measure	After a successfull connection, it should take less than 10 seconds for the application to have updated data

Table 4.18: Web transmission

ID	P3
Source	End user
Stimulus	Application connects to server at a regular interval
Artifact	Finished application
Response	Application transmits all stored data to server
Response measure	After a successfull connection, it should take less than 10 seconds for the application to have sent all stored data

Table 4.19: Reducing handheld application workload

ID	P4
Source	Application
Stimulus	Application recieves data from external sensor
Artifact	Finished application
Response	Data is stored in the handheld database, before it tries to transmit it to the server
Response measure	In normal mode, recieving data from external sensors and sending data to server should happen at regular intervals of X minutes, modifiable by the developers.This to limit amount of data transmission and workload. In Live Data mode, this is difficult to prevent, so user should be made aware of reduced battery life of handheld during Live Data mode. However, sending data to the server should still happen in the same interval of X minutes

Table 4.20: Minimizing data loss

ID	A1
Source	Monitoring nurse End user
Stimulus	Monitoring nurse needs all data recieved at server to be complete
Artifact	Focus on handheld, but entire data transmission chain is involved
Response	The application should have an uptime of 99.999%
Response measure	For this application, an uptime of 99.999% refers more to data retention then actual time. Also, the handheld application does not work alone, meaning the uptime refers to the whole chain as one. To minimize data loss, data is stored on each device (chest unit, handheld, server) until it is sure the data has been transmitted to the next step. This way, theoretically two out of three devices can be down at any given time, with a low risk of losing any recorded data.

Chapter 5

Architectural Description

This chapter introduces the software architecture work that was done prior to the first sprint. In the following discussion, we adhere to the definition of software architecture given in [31]: "The software architecture of a program or computing system is the structure of structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships between them." The purpose of this document is to describe the architecture of our system in a way that is both useful to ourselves and to other interested parties.

This chapter consist of these sections:

- **5.1 Architectural drivers**
  This section will describe the main architectural drivers for our system. These drivers are closely tied to the specifications and wishes of the customer and the requirements identified in chapter 4.
- **5.2 Architectural description**
  This section will contain the system architecture from several different viewpoints, which aims to describe the system in different ways useful to different people.
- **5.3 Architectural rationale**
  This section will explain the different choices we have made with regards to architectural decisions.

5.1 Architectural drivers

The main drivers for our architectural decisions are operating system, functional requirements, extensibility and standards. Especially the operating system and functional requirements have had a significant impact on our decision making during the planning of the architecture.

The operating system for which we are writing this application is the Google Android mobile operating system. It is a java-based system that has some important differences from other operating systems; an non-deterministic life-cycle for applications (more about that here 3.4.1), limited storage space and limited resources for both computational power and battery power. Especially the non-deterministic life-cycle will have to be taken into special consideration alongside the demands for a class 5 system (details can be found in chapter 3).

The functional requirements states that the application has to be able to perform its job without interaction from the user; in other words it needs to work autonomously. This fact will also have significant impact on how we design our system architecture. The main function of the system is to be a bridge between a bluetooth device and a webserver in a health system chain, so redundancy is important to take into consideration in each step in the chain.

That the system is currently being viewed as a prototype by our customer also impacts decision making in the way that the system needs to be easily modified to comply with new bluetooth devices and versions of the bluetooth protocol.

The target consumer group for this system are US military veterans who may or may not be suffering from disabilities preventing them from using a system to its full potential if not taken into consideration. Especially poor eyesight is something we took into consideration when deciding how to lay out the graphical user interface and combining it with the Android look-and-feel and retaining consistency to the Android way of doing things.

5.2 Architectural description

This section will contain a selection of different viewpoints aimed at different audiences:

- **Architecture overview:**
  This view provides a simplified version of the structure of our system, with the most important components and how they are related to each other, and is targeted at people who want a quick overview of the system.

- **Sequence diagram:**
  This view is a basic overview of the call sequence which makes up the primary function of our system and is targeted at people who aim to understand the internal workings of our system.

- **Package diagram:**
  This view contains the package structure of our system and is intended for developers.

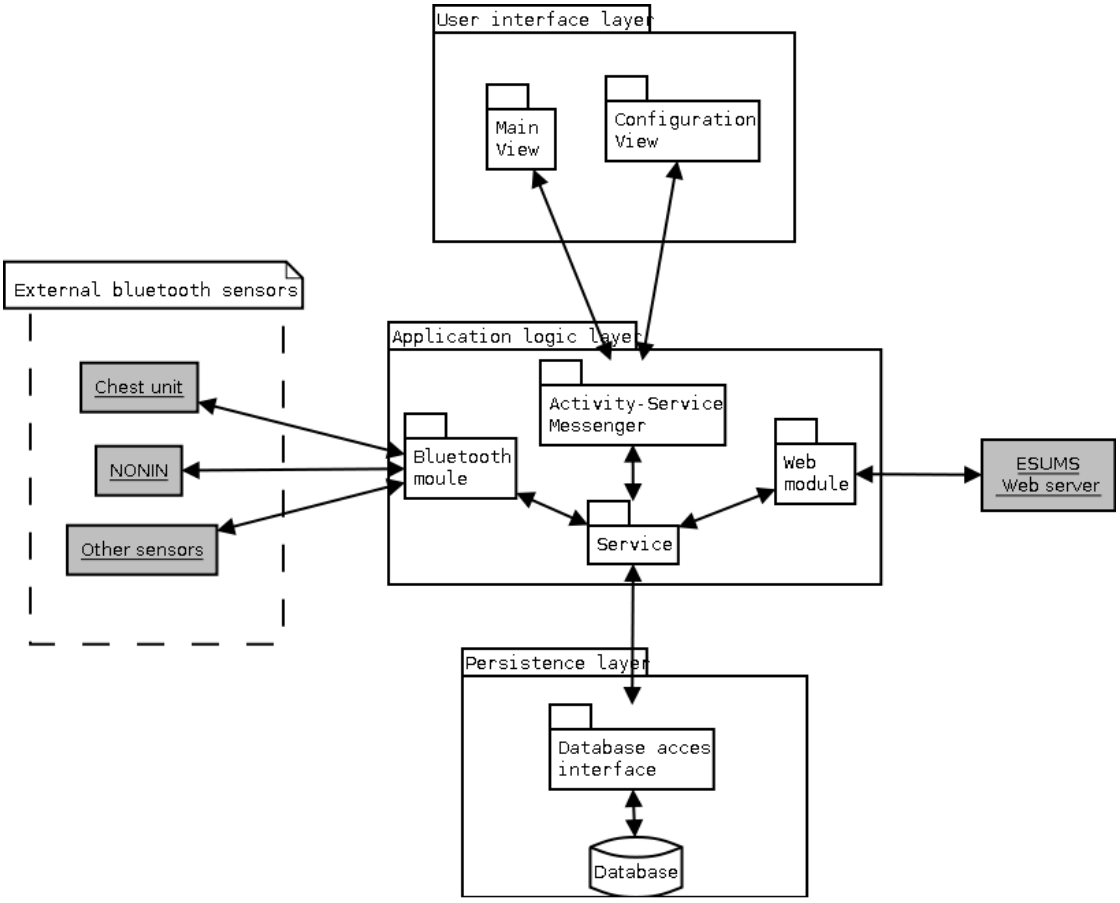

Figure 5.1: Initial architecture for ESUMSDroid, the ESUMS application for Android.

- **Class diagram:**
  This view shows the classes in our system and how they interact with each other and is intended for developers.

- **Database ER diagram:**
  This view is the ER diagram of our database, intended for developers.

5.2.1 Architecture overview

The overall architecture of the ESUMSDroid application is given in figure 5.1. From a high level, the architecture consists of three layers: User interface layer, application logic layer and persistence layer. The three layer model for architecture is one of the most widely used multi-tier architectural models in software

development[32]. Using this architecture for our system ties nicely into the goal of having a largely automated system because communication between layers is only done between neighbouring layers. By putting all the main functionality in a single layer we achieve this goal and we can easily restrict all cross-layer communication to a single point of communication. Each layer is described in more detail below.

User interface (UI) layer The user interface layer is the part of the system that interacts with the user. It displays information to the user and allows the user to issue commands to the system.

Application logic layer The application logic layer is in charge of all system functionality. It controls all operations that take place without the user's interference, such as receiving measurement data from connected devices and sending such data to the ESUMS web server. It also translates commands from the user interface layer into system operations. The application logic layer in our architecture consists of three main modules: The Bluetooth module, the web communications module and the service. The service is the mediator that ties all the parts of the system together, as illustrated in 5.1. This layer has been planned to be as modular as possible, with each module offering a very specific set of services (more detail about this in figure 5.2).

Persistence layer The persistence layer consists of a database, as well as an interface for accessing the database. The service of the application logic layer can initiate retrieval and storage of data through this interface.

5.2.2 Sequence diagram

The diagram in figure 5.2 will show the call sequence of the main functions of ESUMSDroid. In accordance with the requirement specification found in chapter 4, the main functions have been identified as:

- **Fetch data:**
  Fetch data from all connected devices and store the data in the database for temporary persistence while the program completes its cycle.

- **Display data:**
  If the user interface is active display the data to the user. The user interface should also show the status of the main components in the system, which are the ESUMS server connection, the battery level of the chest unit and the connection status of the bluetooth link.

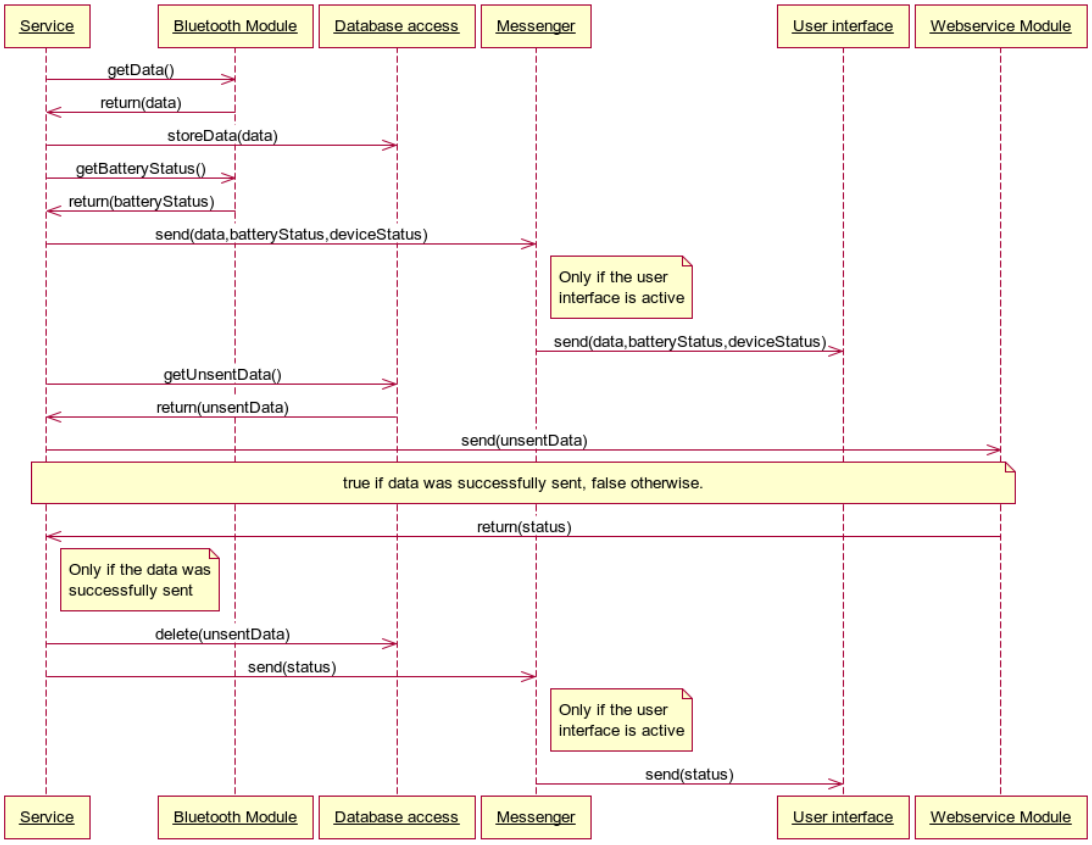

Figure 5.2: Sequence diagram over the primary function of ESUMSDroid.

- **Send data:**
  Send unsent data to the ESUMS server and delete successfully sent data from the database.

The different modules/classes in figure 5.2 have been named in such a way as to retain consistency with the overview architecture (see figure 5.1).

5.2.3 Package diagram

The diagram in figure 5.3 shows the package structure of the source code for the system. Combined with the Android specific files and folders[1] this makes up the code-base of our application. This structuring was constructed with the architecture overview (see figure 5.1) in mind to ensure consistency across views.

[1]The res/ and gen/ folders and manifest file

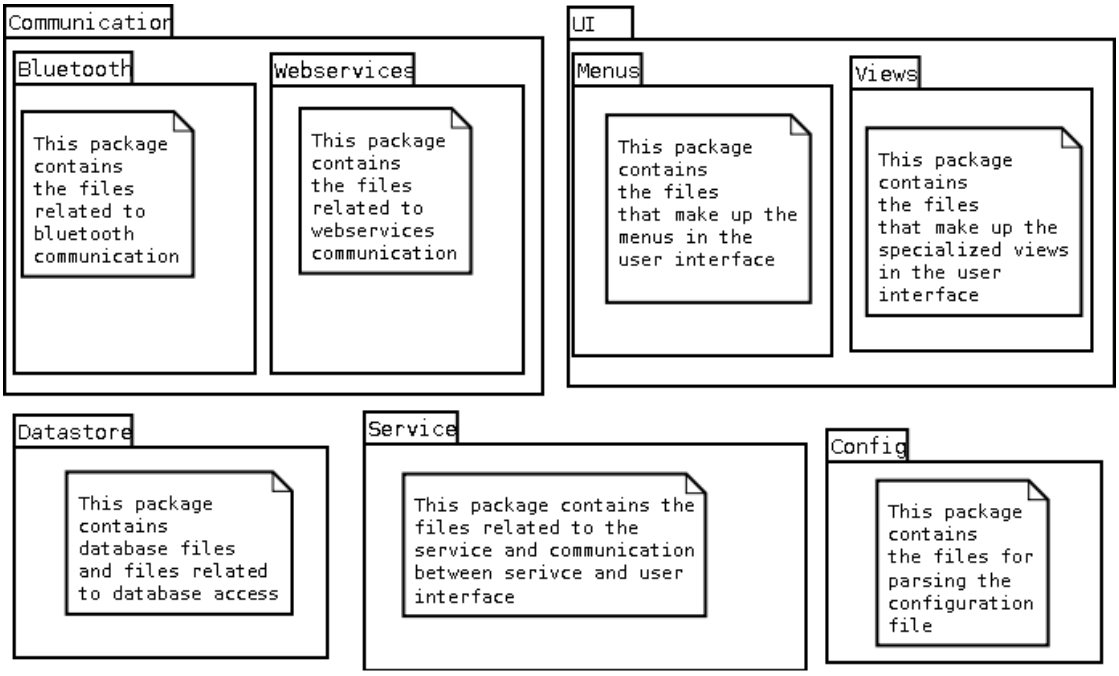

Figure 5.3: Package diagram for ESUMSDroid

This diagram will help the developers to locate code related to a specific purpose and module.

5.2.4 Class diagram

The diagram in figure 5.4 contains the most important classes in our system. This view is not intended to described the entire system in full detail, but to provide an overview of some of the important methods and attributes for the classes included in the view and is likely to change during implementation[2]. The intention behind this diagram is to give developers some starting points and to ensure that everyone is on the same page structure wise. One key point go gather from this view is the use of an abstract class for the bluetooth devices, which greatly reduces the amount of work required to add new sensors to the system.

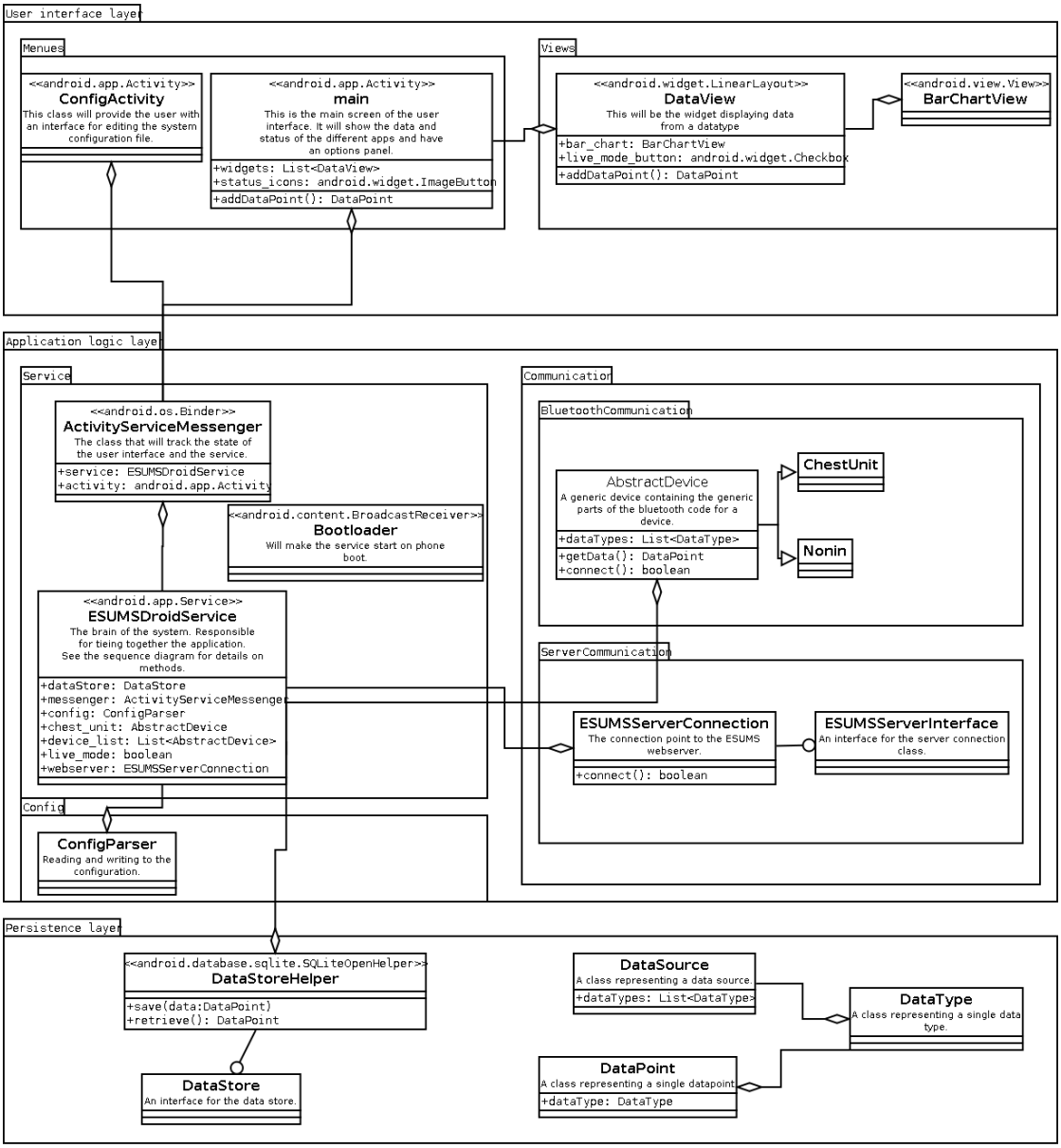

Figure 5.4: Class diagram for ESUMSDroid

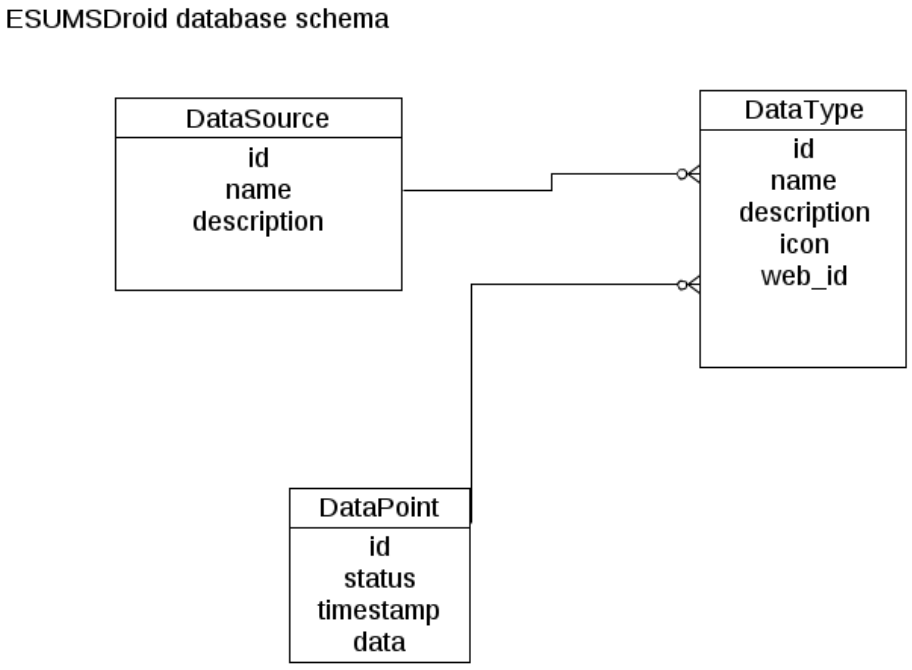

Figure 5.5: Database ER diagram of the ESUMSDroid database.

5.2.5 Database ER diagram

The diagram in figure 5.5 is intended for developers who want details on how the database in ESUMSDroid is structured [3].

5.3 Architectural rationale

We chose a layered architecture because it greatly benefits division of functionality between layers. With one of the main goals being a fully automated system, requiring little to no input from the user, the ability to put all the main functionality into one layer and restrict communication with the other layers to a single class would make this easy to achieve. The division of the functionality in this way also means that modifying our system would become easier as the layers are

[2] An updated and finalized version of this diagram can be found in appendix B.

[3] This diagram has been changed to comply with the changes made to the database in sprint 3.

largely independent from each other when all communication goes through one single class.

Since our program is going to be a prototype for a system that has not been completely described and implemented, having high modifiability is key. Apart from having a layered architecture, we achieve high modifiability through component based structuring of the application logic layer. Especially restricting communication specific code to their respective modules helps this greatly.

Since Android have non-deterministic life-cycles for activities, implementing the main part of the program as a service helps ensuring maximum uptime. Unlike the activities, which need to be in the foreground to preform their function, services can run in the background on the handheld, which makes creating an automated program feasible. Using a service as the main control class for the program gives additional advantages by being the only place where the main program cycle (see figure 5.2 for details) is implemented and the only place where certain objects are instantiated (see figure 5.4 for more details).

Making use of the generality of bluetooth communication code by creating an abstract class for external devices is also motivated by the modifiability aspect as well as reducing the work needed to extend the range of supported devices. Implementing functionality this way also means that switching communication protocols (or upgrading the existing protocol) is less daunting.

Having a persistence layer is motivated by the requirement (details in section 4.1.1) that our program must ensure safe transfer of data between a device and the target ESUMS server, given the uncertain nature of handheld to web communication having this functionality means that we can almost guarantee safe transfer by never deleting data we have not gotten confirmation that the ESUMS server have successfully received.

Chapter 6

Sprint 1

This chapter, along with chapter 7 and 8, cover the implementation phase of the project. Sprint 1 is when the development is started. Here we begin to realize the project according to the requirements described in chapter 4.

This chapter will consist of these sections:

- **6.1 Sprint plan:**
  This section will give a description of the plan for the sprint.
- **6.2 Sprint backlog:**
  This section will be describing which parts of the product backlog are planned for this stage and what the goal is to have completed by the end of the sprint.
- **6.3 Architecture:**
  This section will give a detailed description of the architecture work done in this sprint and any changes made to the architectural description in chapter 5.
- **6.4 Design and implementation:**
  This section will give a detailed description of the design and implementation work completed in this sprint. It will also give details on any changes in the implementation from previous sprints.
- **6.5 Testing and results:**
  This section will describe the testing done during the sprint and the results from the testing.
- **6.6 Sprint retrospective:**
  This section sums up all the work done during the sprint, as an expanded version of the backlog table found in the backlog section with time spent on each task in this sprint. This section will also illustrate the work done as a

burndown chart (see section 3.1.2 for details on this particular part of the scrum process). An evaluation of the process, so far, will also be found in this section.

6.1 Sprint plan

The main purpose of this sprint is mostly to get the development started. The sprint planning meeting was held on Monday 27 September. We had already decided on an overall architecture before this (see chapter 5). The architecture consists mainly of three layers (user interface, object/logic and persistence), and in this first sprint, we intend to implement some basic functionality in all three layers and get them to work together. Some work on the user interface has already been done (before the sprint), as we wanted to visualize early on how the patient is going to interact with the system. Since we are still lacking some information regarding details of the Bluetooth communication with the chest unit, and communication with Web Services, we have decided to allocate implementation of the communication modules to the next sprint. (We can only hope that the customer will have the complete communication protocols ready by then.)

Since Stian has the most experience with Android development, he will be responsible for implementing the service that essentially ties the various other modules together. Dag Øyvind also has some experience with Android and XML, so he will be in charge of user interface work. Both Robin and Øystein have quite a bit of experience with databases, but ystein already has rather a lot of responsibility regarding project management and being main editor for the final report, so Robin will be in charge of the database module implementation for now. The idea here is that the people who have the most experience with the technology will do most of the development work in the beginning, to ensure that we get a solid fundament from the beginning. They can then outsource parts of the programming work to others in the group (possibly during this sprint), as it becomes clear where the tasks can be divided (eg. in the form of method stubs that need to be implemented). This will ease the introduction to Android for those of us who have no previous experience with it. Those who are doing the initial programming may find need for refining parts of the architecture, and they definitely will need to do design work. Whenever they want to change something that might affect the other modules being developed, they need to communicate this to the responsible developer of said module. Everyone will be frequently updated on progress and important changes in scrum meetings several times each week. Having only a few people doing design and implementation in the beginning will help ensure architectural coherence and design consistency throughout the system.

Meanwhile, Thomas and Øystein will be working on writing out the project plan, requirements specification and sprint plan in a form that will be suitable for the final report. The report is after all very important for our final grade, and we want to keep working on it continuously throughout the project, so as to avoid stress at the end. Until it is clear where they can contribute best to the development process, ystein, Thomas and Yushan will also spend some time on self study to become more comfortable with Android.

6.2 Sprint backlog

The sprint backlog is given in table 6.1. We have selected 16 tasks from the product backlog to include in the sprint backlog: Seven user interface tasks, four storage/database tasks and five tasks related to the Android service that ties the whole application together. Implementation of these tasks does not really implement any useful functionality as of yet, i.e.. they do not fully implement any of the functional requirements given in section 4.1.3. This is quite untypical for Scrum, but we are forced to do it this way because we lack the necessary documentation to start implementing any tasks related to Bluetooth or Web server communication (see section 3.1.3).

Table 6.1: The sprint backlog for sprint 1

ID	**Task**	**Responsible**	**Time estimate (h:m)**
UI1	The UI should be able to enable/disable live data from BT units.	Dag Øyvind	1:00
UI2	The UI should be able to show recent data from BT units.	Dag Øyvind	2:00
UI3	The UI should be able to show status of BT connection, Web connection and Battery status for connected BT units.	Dag Øyvind	1:00
UI4	The UI should be able to show recent data from other units.	Dag Øyvind	0:30
UI5	The interface should be able to show a list of other sensors.	Dag Øyvind	1:00
UI6	The UI should be able to show configuration options for administrators	Dag Øyvind	2:00

Continued on next page

Table 6.1 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)
UI7	It should be possible to initiate connection retries from the UI.	Dag Øyvind	1:00
DB1	The database should offer an interface to store data received from the chest unit by the service.	Robin	8:00
DB2	The database should be able to retrieve data-points which have not yet been sent to the server.	Robin	2:00
DB3	The database should be able to store data for a set period after receiving from chest unit. The storage period is part of the configuration done by the administrator.	Robin	2:00
DB4	The database should be able to delete messages which have been verified as received by the web server.	Robin	2:00
S1	The service should be able to switch from Live Mode to Normal Mode	Stian	5:00
S2	The service should be able to switch from Normal Mode to Live Mode	Stian	5:00
S3	The service should be able to prompt the web module to send data to the ESUMS server	Stian	2:00
S5	The service should be able to receive success notification from web module and prompt the database to delete successfully sent messages.	Stian	2:00
S6	The service should be able to relay a request for Bluetooth device data from the user interface to the database.	Stian	3:00
SUM			**39:30**

The total time estimate for all the tasks in this sprint is just below 40 hours. This might seem very low, considering that we have 288 hours (24 hours times 2 weeks times 6 team members) available for the whole sprint. There are several reasons for this low number:

1. There are no more tasks that we can implement at this point, because we lack the necessary documentation from the customer.

2. We expect that quite a bit of studying will be necessary alongside the actual development, as Android is something new for most of us.

3. We want to work on the report continuously in parallel with the development work, so a lot of time will go into this.

6.3 Architecture

The software architecture used throughout this sprint is essentially the same as that described in chapter 5. We include this section mostly to keep a uniform chapter structure for the sprint chapters.

6.4 Design and implementation

As previously mentioned, we did not include any tasks related to the Bluetooth and Web service modules during this sprint. Most of the work was contained within the central "column" of the architecture diagram given in figure 5.1. We will describe each layer separately.

6.4.1 User interface layer

Before the sprint started we had created a few sketches and thrown around some ideas about how the UI should be laid out. During a meeting with the customer we discussed these ideas, and chose a final layout.

The UI has one main view, from which most actions should be selectable (We also have a configuration screen, but this is available only to administrators, through password protection. It is hugely uninteresting, consisting mainly of text-boxes and sliders, and so we'll leave out a detailed description). The intention of using only one screen for all functionality was to keep things simple and understandable. We wished to make the application as transparent as possible for the user.

The main UI has two major sections. At the top of the screen we show the status of devices and the web service. Any problems with connectivity or low battery are reported in this section. Below is the main data views. This section displays a graphical representation of the latest data received from each datasource.

During this sprint we wanted to get the layout in place, without implementing much of the functionality of the UI. In Android, layouts are specified in XML-files, and most of the work was related to finding the best widgets available for the task at hand. We also took some care to make the UI configurable, with regards to internationalization, and to make it receptive to adding functionality later on. We also had to implement a custom data-widget, which turned out to be quite a bit of work. This widget had to display a lot of data in a limited amount of space, and we decided to use a bar-chart. This works very well, as the user has a clear indication of trends in the data, for instance seeing an upward slope when working out, or a flat trough when relaxed. Bar-charts pack information very densely, without becoming confusing. We had to make sure the widget was able to display this data on a variety of screen sizes and allow it to be configurable to show different kinds of data. We'll leave the details out, but this involved normalizing the data to type-specific boundaries, and then scaling it to fit on different screens.

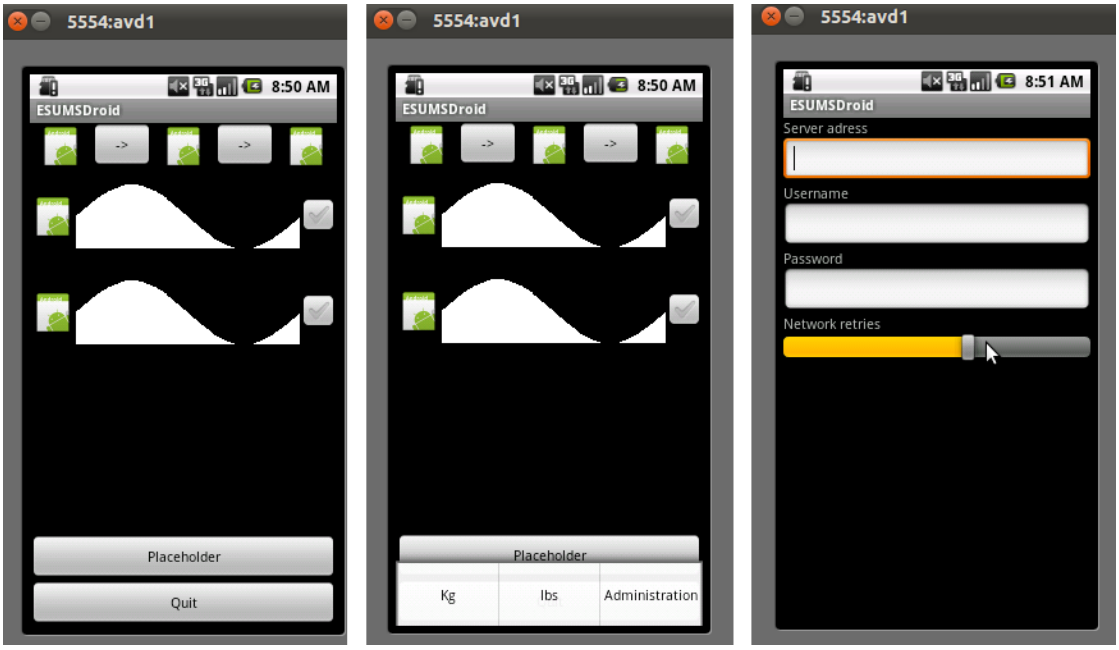

Figure 6.1: The UI as it was during sprint 1. From left to right:
-The main screen of the application, with placeholder icons, and testing data.
-The main screen showing the application menu at the bottom.
-The configuration screen of the application.

6.4.2 Application logic layer

As described in the architecture chapter, the service is mostly the glue which mediates data-requests between the different modules in the application. It is also responsible for scheduling tasks when the user interface is inactive, as the application can be considered "always-on".

With this in mind, we set up interfaces between the service and the different components of the application. For the user interface this turned out to be more difficult than expected, as we have no direct control of the lifetime of an Activity in Android. This is a limitation of Android, and could not be helped. Android has three features to allow communication between a service and other components of an application, which are covered in section 3.4.1. Messengers, which allow two-way communication, but in a rather backwards manner, was chosen to be the method of communication. All communication via messengers are implemented using callbacks. This meant that the user interface had to be implemented with asynchrony in mind. Since interfacing the user interface and service was planned for sprint 2, we forged ahead with this solution.

The service also implements a fairly powerful scheduler for tasks which must be completed at regular intervals, such as sending data to the web service.

6.4.3 Data persistence layer

The data persistence layer was thankfully fairly straight-forward to implement, as Android supports SQL through the sqlite3 library. Based on the criteria for persistence, searching and thread-safety we had, sqlite3 was the clear choice. Selecting the appropriate database schema for the task at hand was easy once the alternatives became obvious. Basically the choices were between a schema that was extensible and efficient, and a few alternatives that had various degrees of extensibility and efficiency, but did not match the obvious choice for either.

We implemented the data storage module as a collection of classes that represent tables. An instance of a class then corresponds to a row. This means that no other component of the application need to know any SQL or indeed that we are even using SQL, ideally making the application more loosely coupled. This component was the easiest one of the lot to test properly, and as such we decided at an early stage that we wanted to use unit tests for this one. While we did not finish them for this sprint, we verified that all the methods ran, and declared the module ready for usage.

At this point, the data storage supported, amongst other things

- Selection of unsent data
- Marking data as sent

- Selection of data based on which sensor or type it has
- Associating names and descriptions with sensors
- Saving or updating data in bulks

An ER-diagram showing the details is available in the architecture overview 5.5.

6.5 Testing and results

This section describes the our testing methodology and test performed, as well as the deliverable product increment, as scrum calls the application.

6.5.1 Testing

During the initial planning we had hoped to use some methodology from TDD (Test Driven Development), where all major classes and all modules have corresponding unit tests. We believed this would give us an edge in acceptance testing, and allow us to work more efficiently. However, it quickly became apparent that due to the high amount of external components (such as Bluetooth devices and the web service), we would need an extraordinarily high number of complex mock objects, essentially reimplementing all external components. Due to this lack of foresight, we decided to revise our testing methodology, and postpone systematic testing until such a time as we had the external components available.

Some testing was still performed. The data persistence layer was completely independent of other components, and we started planning unit testing for it. These were not finished by the end of the sprint, nor planned at the start, and as such are not mentioned in detail in this chapter. We will revisit the topic in Sprint 2.

For other components we designed specific runs of the application, which were designed to test a single feature of the application, and to discover any bugs which may have been present. With carefully placed logging in the application we could determine that each component handled the feature in the correct way, and communicated correctly with other components.

An example of such a run was handling reconnection to a bluetooth device. We started by generating a message from the service to the UI that a device had been disconnected, and checked that the correct message was shown to the user. This was then followed by a call to the bluetooth module, which generated a new message that the device had been reconnected. Once again the UI was notified of the new connection status, and we checked that the correct messages were displayed.

These test were cumbersome, and had to be manually run, which was time-consuming. They were only run once a feature had been deemed complete enough that we hoped not to touch that section of the code again. We believe the time spent was worth it, as we had no other means of testing, and it allowed us to proceed on new tasks with the confidence of having completed the previous task to specification.

6.5.2 Results

At the end of the sprint, we had fully implemented the data storage layer, and had the layout of the UI done. The service part of the application logic layer was coming along nicely, but there was still quite a bit of integration to be done, with regards to the Bluetooth communication module and the web service.

In all, we had a barebones application which could run, but did not contain any useful functionality. Our goal was to lay the foundation to start implementing the required functions, and to assert that our architecture was feasible on the Android platform. Both these goals were successful, and as such we were happy with the work we had accomplished.

6.6 Sprint retrospective

We completed all the tasks in the sprint backlog. The sprint backlog, updated with an additional column for time spent per task, is given in table 6.2. As can be seen from the table, the time estimates for the various tasks were mostly accurate, although there are some notable anomalies. The single biggest such was in requirement UI2, where two members of the team had worked on the same requirement. This was perhaps the biggest failure of communication the team experienced, and after it was discovered several new protocols were put in place to avoid such an incident in the future. Amongst these were stricter logging of hours, and improved communication of work packets during daily scrums.

Another source of anomalies in the backlogs is that many items are interconnected, so that when one is nearing completion, several others are also nearly done. This was not something we had anticipated, and made estimating tasks much harder. This is true of all the burndown charts and backlog items for each sprint.

The final source of anomalies lies in automation. The time spent was summed automatically, and relied on very careful logging of which hours went into which tasks. This was not always done carefully enough, and as a result the numbers in the Time spent column are not always 100% correct.

Figure 6.2 shows the burndown chart of sprint 1. A burndown chart is a useful tool for tracking the progress of task accomplishment. At the start of a sprint, the team estimates the time required for each task. This gives us a good starting point for planning when each task must be worked on. Each day, team members update the estimates of their assigned task. This doesn't necessarily mean the estimate must go down, as our chart shows on the 28th of September. As the sprint progresses, we expect to see the graph trend downwards as tasks are completed, and hopefully reach zero on the last day of the sprint. For this sprint, we see that this was indeed accomplished. This proved to be a useful tool for the team, as we could quickly determine whether we were on track.

The tool was maintained in a spreadsheet on google-docs, with some clever formulae keeping track of hours spent. As mentioned before, this was an automated process, and aside from the initial time spent on creating the spreadsheet, did not cost us anything in terms of time spent. Being an automated process it was still not perfect, but a substantial boost for productivity nonetheless, and we are very happy we chose to use it.

Table 6.2: Sprint backlog for sprint 1 with time spent per task

ID	**Task**	**Responsible**	**Time estimate (h:m)**	**Time spent (h:m)**	**Time remaining (h:m)**
UI1	The UI should be able to enable/disable live data from BT units.	Dag Øyvind	1:00	0:50	0:00
UI2	The UI should be able to show recent data from BT units.	Dag Øyvind	2:00	9:10	0:00
UI3	The UI should be able to show status of BT connection, Web connection and Battery status for connected BT units.	Dag Øyvind	1:00	0:00	0:00
UI4	The UI should be able to show recent data from other units.	Dag Øyvind	0:30	1:30	0:00
UI5	The interface should be able to show a list of other sensors.	Dag Øyvind	1:00	0:00	0:00
UI6	The UI should be able to show configuration options for administrators	Dag Øyvind	2:00	0:00	0:00
UI7	It should be possible to initiate connection retries from the UI.	Dag Øyvind	1:00	0:00	0:00

Continued on next page

Table 6.2 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)	Time spent (h:m)	Time remaining (h:m)
DB1	The database should offer an interface to store data received from the chest unit by the service.	Robin	8:00	9:30	0:00
DB2	The database should be able to retrieve data-points which have not yet been sent to the server.	Robin	2:00	1:45	0:00
DB3	The database should be able to store data for a set period after receiving from chest unit. The storage period is part of the configuration done by the administrator.	Robin	2:00	1:00	0:00
DB4	The database should be able to delete messages which have been verified as received by the web server.	Robin	2:00	1:00	0:00
S1	The service should be able to switch from Live Mode to Normal Mode	Stian	5:00	5:00	0:00
S2	The service should be able to switch from Normal Mode to Live Mode	Stian	5:00	4:00	0:00
S3	The service should be able to prompt the web module to send data to the ESUMS server	Stian	2:00	3:00	0:00
S5	The service should be able to receive success notification from web module and prompt the database to delete successfully sent messages.	Stian	2:00	2:00	0:00
S6	The service should be able to relay a request for Bluetooth device data from the user interface to the database.	Stian	3:00	2:00	0:00
SUM			**39:30**	**40:45**	**0:00**

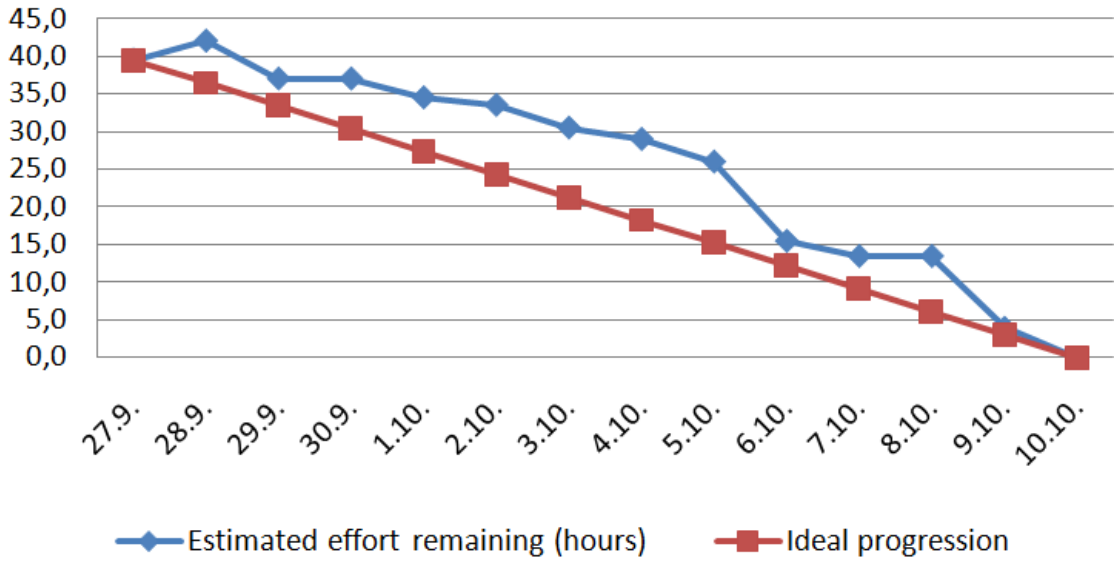

Figure 6.2: Burndown chart for sprint 1

Chapter 7

Sprint 2

This chapter covers sprint 2 out of 3. At this stage of the project, work on the modules we could not begin in sprint 1 should be started on. This mainly means the web service module and the bluetooth module, which are needed to fulfill the main functionality of the system.

This chapter will consist of these sections:

- **7.1 Sprint plan:**
  This section will give a description of the plan for the sprint.
- **7.2 Sprint backlog:**
  This section will be describing which parts of the product backlog are planned for this stage and what the goal is to have completed by the end of the sprint.
- **7.3 Architecture:**
  This section will give a detailed description of the architecture work done in this sprint and any changes made to the architectural description in chapter 4.
- **7.4 Design and implementation:**
  This section will give a detailed description of the design and implementation work completed in this sprint. It will also give details on any changes in the implementation from previous sprints.
- **7.5 Testing and results:**
  This section will describe the testing done during the sprint and the results from the testing.
- **7.6 Sprint retrospective:**
  This section sums up all the work done during the sprint, as an expanded version of the backlog table found in the backlog section with time spent on

each task in this sprint. This section will also illustrate the work done as a burndown chart (see section 3.1.2 for details on this particular part of the scrum process). An evaluation of the process, so far, will also be found in this section.

7.1 Sprint plan

In this sprint, we intend to implement most of the remaining functional requirements in the product backlog. Mostly all tasks from Sprint 1 were completed (with the exception of a few small details that had to be postponed due to dependency issues), so we will not have to transfer much work from sprint 1 to sprint 2.

Specifically, this sprint will concern the communication aspects of the handheld client, that is communication with the chest unit and other sensors via Bluetooth, as well as communication with the ESUMS web server over the Internet. When the communication framework is in place, it remains to make sure all the required functionality is accessible from the GUI and works as specified. Using well defined interfaces between the software modules, we are able to work on this independently of the work on the actual communication (Bluetooth and Web) modules.

Last sprint, we ran out of tasks to do before the allotted time was spent. To avoid running into this situation again, we intentionally add many more backlog items than we believe we have time to finish this sprint. Some of the tasks will be impossible to complete before we obtain the actual devices that they concern, but as we do not yet have any expected time of arrival for the bluetooth sensors, we add these items to the backlog anyway, just in case we should receive them. This will probably lead to some backlog items having been implemented, but not tested. We anticipate that there will be many uncompleted items this sprint, possibly making for a burndown chart that looks deceptively bad. We believe that these changes in the process will increase the amount of work we get done.

7.2 Sprint backlog

Table 7.1: The sprint backlog for sprint 2

ID	**Task**	**Responsible**	**Time estimate (h:m)**
T1	Set up unit testing framework	Robin	7:00

Continued on next page

Table 7.1 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)
T2	Data store tests	Robin	4:00
BT1	The Bluetooth module should be able to receive a prompt for a search for paired Bluetooth devices in range.	Dag Øyvind	1:00
BT2	The Bluetooth module should be able to prompt the Handheld to search for bluetooth devices in range.	Dag Øyvind	1:00
BT3	The Bluetooth module should be able to prompt retries for a set number of times if no units are found.	Dag Øyvind	1:00
BT4	The Bluetooth module should be able to tell the service that no units were found.	Dag Øyvind	0:30
BT5	The Bluetooth module should be able to connect to a Bluetooth device.	Dag Øyvind	2:00
BT6	The Bluetooth module should retry a set number of times if no connection is made.	Dag Øyvind	0:30
BT7	The Bluetooth module should be able to tell the service that no connection was made.	Dag Øyvind	0:30
BT8	The Bluetooth module should be able to request data from Bluetooth device.	Dag Øyvind	0:30
BT9	The Bluetooth module should be able to retry the request for data from Bluetooth device a set number of times, and report failure to the server if no connection is made.	Dag Øyvind	0:30
BT10	The Bluetooth module should be able to read battery status of Chest unit from recently received data and report it to the service.	Dag Øyvind	0:30
BT11	The Bluetooth module should be able to read timestamps from the sensors, or automatically time-stamp them at time of receipt if no time-stamp is available.	Dag Øyvind	0:30
BT12	The Bluetooth module should be able to tell the service that a connection has been lost due to being out of range.	Dag Øyvind	0:30

Continued on next page

Table 7.1 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)
W0	Set up ESUMS web server	Dag Øyvind	2:00
W1	The web module should be able to receive prompt to connect to the ESUMS server.	Thomas	1:00
W2	The web module should be able to connect to the ESUMS server.	Thomas	1:00
W3	If the web module cannot connect to the server it should retry a set number of times.	Thomas	0:30
W4	If the web module cannot connect after the set number of times it should notify the user, and offer an option to retry.	Thomas	0:30
W5	The web module should be able to authorize with the ESUMS server.	Thomas	1:00
W6	If the web module cannot authorize with the server it should notify the user, and have him contact an administrator (username or password are incorrectly set).	Thomas	0:30
W7	The web module should be able to send data to the web server, as specified by the WSDL.	Thomas	2:30
W8	The web module should be able to receive responses which verify the data has been transmitted and notify the service of the success.	Thomas	1:00
W9	The web module should be able to resend data a set number of times or, if the retries fails, notify the service of the failure.	Thomas	1:00
UI1	The user interface should be able to select/deselect Live Mode.	Dag Øyvind	2:00
UI2	The user interface should be able to display recent data from Bluetooth device or, if no data exists, notify the user.	Dag Øyvind	3:00
UI6	It should be possible to change congura-tion le details via the user interface by entering an admin password	Dag Øyvind	3:00

Continued on next page

Table 7.1 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)
UI7	It should be possible to click on the icons for the different statuses in the user interface.	Dag Øyvind	2:00
DB5	The data storage module must have a method for retrieving data that was recently stored.	Robin	0:30
S4	The service should be able to relay connection and authorization status notifications from the web module to the user interface.	Stian	2:00
S7	The service should be able to detect that the Chest unit battery is low.	Stian	3:00
S8	The service should be able to tell the user interface of the battery status.	Stian	3:00
S9	The service should be able to play a sound to notify the User.	Stian	2:00
S10	The service should be able to notify the User that the Chest unit battery is low.	Stian	2:00
S11	The service should be able to prompt the web module to connect to the ESUMS server.	Stian	2:00
S12	The service should be able to relay a request for a Bluetooth device list from the user interface to the database	Stian	4:00
S13	The service should be able to prompt the Bluetooth module to connect to a Bluetooth device.	Stian	2:00
S14	The service should be able to relay connection status notications from the Bluetooth module to the user interface.	Stian	2:00
S15	The service should be able to relay storage requests from the Bluetooth module to the database.	Stian	2:00
SUM			**65:30**

7.3 Architecture

During this sprint we have kept the overall initial architecture, as described in chapter 5. It was of course pleasant to discover that the effort of our pre-study had paid off, and that our design was sufficient for the application we were creating. We did have to change some of the interfaces between modules, but the changes were minor and did not change the way components interacted, but rather what messages they would pass each other.

7.4 Design and Implementation

7.4.1 User interface layer

The user interface was more or less finished (bar integration with the service) last sprint, and only a couple of requirements carried over from sprint 1 to be finished in this sprint.

The few requirements which had to be transferred from sprint 1 were mostly completed, with the full framework ready and only details to fill in.

Despite this, quite a few hours were spent, as errors were discovered, and fixed. The integration with the service also took longer than expected. We had been unable to test this during sprint 1, as we had no real data with which to perform testing. This was something we had half expected, and therefore were ready to handle.

We also had to change the way the UI and service communicated, as mentioned in the architectural changes. This was due to a limitation in how Android handles UIs. Android runs a separate thread for all UI related calls. This is normal in UI libraries, as one needs a thread to send events between widgets. However, in Android this is the only thread which is allowed to interact with widgets. This presented a problem when the service (which was running in its own thread) wanted to send data to the UI. We had to change the interface between the UI and service, and make some significant changes in the UI code. The issue was resolved without any significant problems, but this did take quite a bit of time. No graphical changes were made to the user interface, so it retained its charming looks from sprint 1 which can be seen in figure 6.1.

7.4.2 Application logic layer

This is where most of our efforts during sprint 2 were directed. The goal of this sprint was to get the web-server module and the bluetooth module operational and to finish the rest of the service. Together these parts creates the vertical row

through the center of figure 5.1 and is where the main functionality of our system lies.

Service

Some significant changes were made to the way the service would interact with the user interface, as we scrapped the work done with Messengers and switched to using a Binder to act as a mediator between the service and the user interface, so that neither would have to be aware of the state of the other.

The operation cycle of the service was completed, which means that the tasks that the service carries out each cycle were completed. This cycle consists of three main tasks with various subtasks:

- Get new data from devices connected to the phone and report any failures to the user interface.
- Save the data in the database and send the new data to the user interface.
- Send the new data to the web-server and remove the successfully sent data from the database.

At this point we were still regarding each part of the system as a standalone module, with integration planned for the last sprint.

Bluetooth module

During this sprint, the NONIN sensor was made available to the group, and work went into making a sensible interface between general bluetooth devices and the rest of the application. Most of the work needed for the chest unit to function properly was also done here. A general bluetooth device class was created, which connects to any bluetooth device, and listens for input. The code that was specific for NONIN was related to parsing the data it sends. When NONIN was tested, it uncovered a few bugs in the rest of the application, namely the data persistence layer. Aside from these road-bumps, most of the work went smoothly. Initially we had thought to be using a Java interface to define devices, but we discovered that it worked better with an abstract class that implements all the functionality that will be the same for devices, so we modified our design slightly here.

The abstract class for devices was also prepared for the chest unit, so that we would be able to work quickly with it for the few days we would have access to it. This mostly means that necessary logging and debugging constructs were in place. We had initially planned to use a threaded approach for devices, as bluetooth connection attempts take about 12 seconds to time out, but we decided against a fully threaded approach because of the difficulties of propagating errors properly

from a different thread, and went for a hybrid approach, where the connection attempt is threaded, but the rest of the life-cycle of a device is not.

The customer also informed us that the ESUMS team had decided to move away from the standards (mentioned in section 3.5) related to the bluetooth device communication, and especially moving away from IEEE 11073 gave us an easier time implementing the bluetooth communication.

Web-server module

Work on the web service module started during this sprint. The purpose of the web service module is to remotely connect to the ESUMS server and transmit datapoints collected from the external sensors. It also needs to ensure the data has successfully arrived at the server. The module is based on WSDL and the idea is to implement it using an Android specific version of kSOAP2. The module is designed to respond to requests from the service. These requests involve connecting to the ESUMS server and transmitting datapoints. The module must perform these requests and respond to the service with a status report. Details can be found in appendix B.

During this sprint the structure of the module was created. The necessary class skeletons were created and basic implementation was started. The main functionality of the module were mostly implemented through two methods, listed below. One for connecting to the server and one for sending data to the server. They return the parsed response they get from the server.

- boolean sendData(ArrayList<DataPoint> data)
- boolean connect()

Due to illness this module could not be fully completed or tested during the sprint, as planned, however implementation got started, and some spillover into sprint 3 was not seen as a big problem.

7.4.3 Data persistence layer

Not much work was done to the data persistence layer during this sprint, mostly just fixing various bugs that would appear when used with other parts of the application. Most serious bugs had been weeded out with the unit tests, but a few were uncovered that weren't covered by tests. It also turned out that tracebacks for some types of errors weren't very useful in hunting them down, which caused a few hours of debugging an exception where the source was a method call that was not being invoked outside of the data storage layer.

7.5 Testing and results

In this sprint we created unit-tests for the data persistence layer. These were successful in weeding out a few bugs which had not been noticed during our testing during sprint 1.

The team got two Android-phones from the customer, to perform actual testing on devices. These were:

- HTC Desire
- Samsung I9000 galaxy S

This allowed us to test and profile the application in a real environment, which was invaluable.

A Nonin was also made available to us by the customer, which meant we could now perform some testing in the bluetooth module. These were mostly related to common bluetooth features, such as connecting, reading data (without interpreting it in any way) and handling errors which might occur.

7.6 Sprint retrospective

This sprint was handled somewhat differently than the first. We had uncovered some problems in our process, and wished to address them.

First, we over-allocated tasks. This boosted productivity, as the team could work evenly throughout the sprint. Another reason for doing this was that some tasks would be blocked by not having the external devices available, most notably the chest unit. By over-allocating tasks both these problems were resolved, and we are happy with the decision. This does however mean that the backlog contains unresolved tasks, as can be seen in the burndown chart in figure 7.1 not ending up at zero hours estimated at the end. We consider this an acceptable tradeoff.

If one looks carefully at the backlog items, some seem to be transferred over without having any work done to them. In reality, these tasks were completed in a matter of hours, but the hours were not logged properly. This once again highlights the problem of not having an automated process for logging progress.

At the end of the sprint our application had taken significant steps towards completion. Our webserver-module was not as complete as we had hoped, due to illness, and the bluetooth module was lacking in device specific testing, as we still did not have access to the chest unit. Aside from these problems we were happy with the result.

Table 7.2: Sprint backlog for sprint 2 with time spent per task

ID	**Task**	**Responsible**	**Time estimate (h:m)**	**Time spent (h:m)**	**Time remaining (h:m)**
T1	Set up unit testing framework	Robin	7:00	3:30	4:00
T2	Data store tests	Robin	4:00	2:40	2:00
BT1	The Bluetooth module should be able to receive a prompt for a search for paired Bluetooth devices in range.	Dag Øyvind	1:00	0:00	1:00
BT2	The Bluetooth module should be able to prompt the Handheld to search for bluetooth devices in range.	Dag Øyvind	1:00	0:30	0:00
BT3	The Bluetooth module should be able to prompt retries for a set number of times if no units are found.	Dag Øyvind	1:00	0:30	0:00
BT4	The Bluetooth module should be able to tell the service that no units were found.	Dag Øyvind	0:30	0:00	0:00
BT5	The Bluetooth module should be able to connect to a Bluetooth device.	Dag Øyvind	2:00	1:00	0:00
BT6	The Bluetooth module should retry a set number of times if no connection is made.	Dag Øyvind	0:30	0:00	0:30
BT7	The Bluetooth module should be able to tell the service that no connection was made.	Dag Øyvind	0:30	0:30	0:00
BT8	The Bluetooth module should be able to request data from Bluetooth device.	Dag Øyvind	0:30	1:00	0:00
BT9	The Bluetooth module should be able to retry the request for data from Bluetooth device a set number of times, and report failure to the server if no connection is made.	Dag Øyvind	0:30	0:30	0:00

Continued on next page

Table 7.2 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)	Time spent (h:m)	Time remaining (h:m)
BT10	The Bluetooth module should be able to read battery status of Chest unit from recently received data and report it to the service.	Dag Øyvind	0:30	0:00	0:00
BT11	The Bluetooth module should be able to read timestamps from the sensors, or automatically time-stamp them at time of receipt if no time-stamp is available.	Dag Øyvind	0:30	0:00	0:00
BT12	The Bluetooth module should be able to tell the service that a connection has been lost due to being out of range.	Dag Øyvind	0:30	0:00	0:00
W0	Set up ESUMS web server	Dag Øyvind	2:00	7:30	0:00
W1	The web module should be able to receive prompt to connect to the ESUMS server.	Thomas	1:00	0:00	0:30
W2	The web module should be able to connect to the ESUMS server.	Thomas	1:00	1:00	0:30
W3	If the web module cannot connect to the server it should retry a set number of times.	Thomas	0:30	0:00	0:30
W4	If the web module cannot connect after the set number of times it should notify the user, and offer an option to retry.	Thomas	0:30	0:00	0:30
W5	The web module should be able to authorize with the ESUMS server.	Thomas	1:00	1:00	0:30
W6	If the web module cannot authorize with the server it should notify the user, and have him contact an administrator (username or password are incorrectly set).	Thomas	0:30	0:00	0:30
W7	The web module should be able to send data to the web server, as specified by the WSDL.	Thomas	2:30	1:30	1:00

Continued on next page

Table 7.2 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)	Time spent (h:m)	Time remaining (h:m)
W8	The web module should be able to receive responses which verify the data has been transmitted and notify the service of the success.	Thomas	1:00	0:30	0:30
W9	The web module should be able to resend data a set number of times or, if the retries fails, notify the service of the failure.	Thomas	1:00	0:00	1:00
UI2	The user interface should be able to select/deselect Live Mode.	Dag Øyvind	2:00	1:00	3:00
UI3	The user interface should be able to display recent data from Bluetooth device or, if no data exists, notify the user.	Dag Øyvind	3:00	0:00	1:00
UI6	It should be possible to change configuration details via the user interface by entering an admin password	Dag Øyvind	3:00	3:00	3:00
UI7	It should be possible to click on the icons for the different statuses in the user interface.	Dag Øyvind	2:00	0:00	1:00
DB5	The data storage module must have a method for retrieving data that was recently stored.	Robin	0:30	0:30	0:00
S4	The service should be able to relay connection and authorization status notifications from the web module to the user interface.	Stian	2:00	1:00	1:00
S7	The service should be able to detect that the Chest unit battery is low.	Stian	3:00	2:00	0:00
S8	The service should be able to tell the user interface of the battery status.	Stian	3:00	2:00	0:00
S9	The service should be able to play a sound to notify the User.	Stian	2:00	1:30	0:00
S10	The service should be able to notify the User that the Chest unit battery is low.	Stian	2:00	3:00	0:00

Continued on next page

Table 7.2 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)	Time spent (h:m)	Time remaining (h:m)
S11	The service should be able to prompt the web module to connect to the ESUMS server.	Stian	2:00	2:00	0:00
S12	The service should be able to relay a request for a Bluetooth device list from the user interface to the database	Stian	4:00	2:30	1:00
S13	The service should be able to prompt the Bluetooth module to connect to a Bluetooth device.	Stian	2:00	2:30	0:00
S14	The service should be able to relay connection status notifications from the Bluetooth module to the user interface.	Stian	2:00	2:00	0:00
S15	The service should be able to relay storage requests from the Bluetooth module to the database.	Stian	2:00	2:30	0:00
SUM			**65:30**	**47:10**	**23:00**

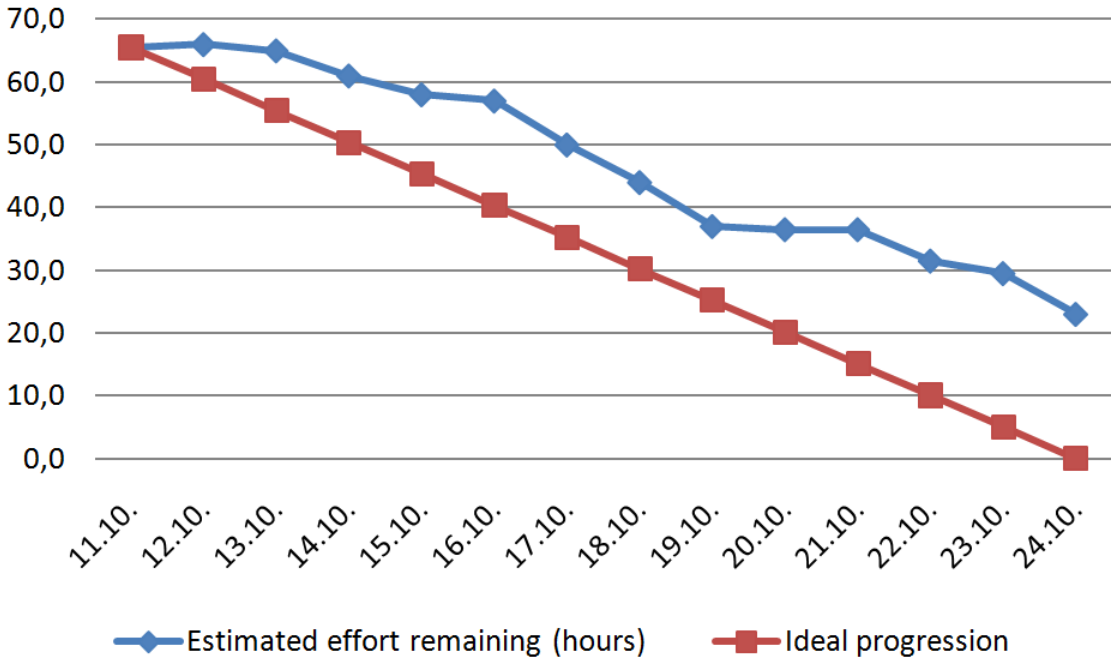

Figure 7.1: Burndown chart for sprint 2

Chapter 8

Sprint 3

Sprint 3 will our final sprint. At the end of this sprint, the application should be functional as described by the requirements. As the application should be completed by the end of the sprint, there will be additional focus on testing. At that point we can hopefully perform complete system tests and ensure everything works together. The whole process, from the chest unit and NONIN, through the handheld application to the ESUMS server, should be working and communicating as planned.

This chapter will consist of these sections:

- **8.1 Sprint plan:**
  This section will give a description of the plan for the sprint.

- **8.2 Sprint backlog:**
  This section will be describing which parts of the product backlog are planned for this stage and what goals are to be completed by the end of the sprint.

- **8.3 Architecture:**
  This section will give a detailed description of the architecture work done in this sprint and any changes made to the architectural description in chapter 4.

- **8.4 Design and implementation:**
  This section will give a detailed description of the design and implementation work completed in this sprint. It will also give details on any changes in the implementation from previous sprints.

- **8.5 Testing and results:**
  This section will describe the testing done during the sprint and the results from the testing.

- **8.6 Sprint retrospective:**
  This section sums up all the work done during the sprint, as an expanded version of the backlog table found in the backlog section with time spent on each task in this sprint. This section will also illustrate the work done as a burndown chart (see section 3.1.2 for details on this particular part of the scrum process). An evaluation of the process up to this point will also be found in this section.

8.1 Sprint plan

As this is the last sprint, the plan is to basically implement everything that still remains. All the backlog items not yet completed have been selected, including those from sprint 2 that were not finished. The main tasks still remaining are the web service module, bluetooth module and the integration of these into the rest of the application. In addition, a few tasks from each of the other modules still remain as well. At the end of the sprint, we have no more time for this project. Therefore the object of this sprint is to finish the handheld application and have a functional prototype ready to hand over to the customer. To evaluate the final product, the testing performed in this sprint will be the system integration testing as well as the acceptance testing.

8.2 Sprint backlog

Table 8.1: The sprint backlog for sprint 3

ID	**Task**	**Responsible**	**Time estimate (h:m)**
T1	Set up unit testing framework	Robin	4:00
T2	Data store tests	Robin	2:00
BT1	The Bluetooth module should be able to receive a prompt for a search for paired Bluetooth devices in range.	Dag yvind	1:00
BT6	The Bluetooth module should retry a set number of times if no connection is made.	Dag Øyvind	0:30
W1	The web module should be able to receive a prompt to connect to the ESUMS server.	Thomas	0:30

Continued on next page

Table 8.1 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)
W2	The web module should be able to connect to the ESUMS server.	Thomas	0:30
W3	If the web module cannot connect to the server it should retry a set number of times.	Thomas	0:30
W4	If the web module cannot connect after the set number of times it should notify the user, and offer an option to retry.	Thomas	0:30
W5	The web module should be able to authorize with the ESUMS server.	Thomas	0:30
W6	If the web module cannot authorize with the server it should notify the user, and have him contact an administrator (username or password are incorrectly set).	Thomas	0:30
W7	The web module should be able to send data to the web server, as specified by the WSDL.	Thomas	1:00
W8	The web module should be able to receive responses which verify the data has been transmitted and notify the service of the success.	Thomas	0:30
W9	The web module should be able to resend data a set number of times, or, if the retries fail, notify the service of the failure.	Thomas	1:00
UI1	The user interface should be able to select/deselect Live Mode.	Dag Øyvind	3:00
UI2	The user interface should be able to display recent data from Bluetooth device or, if no data exists, notify the user.	Dag Øyvind	1:00
UI6	It should be possible to change configuration details via the user interface by entering an admin password	Dag Øyvind	3:00
UI7	It should be possible to click on the icons for the different statuses in the user interface.	Dag Øyvind	1:00
UI8	It should be possible to click on each graph to open a fullscreen view of that graph.	Dag Øyvind	3:00

Continued on next page

Table 8.1 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)
UI9	The range of the graphs' axes should adjust dynamically to span the data fully	Dag Øyvind	1:00
UI10	Add thresholds for normal range of the various measurement data.	Dag Øyvind	2:00
11	The service and the user interface should be able to talk to each other and exchange necessary messages and data.	Stian	15:00
I2	The service and the Bluetooth module should be able to talk to each other and exchange necessary information and objects.	Robin	15:00
I3	The service and the storage module should be able to talk to each other and exchange necessary information and objects.	Stian	10:00
I4	The service and the web module should be able to talk to each other and exchange necessary information and objects.	Robin	10:00
S4	The service should be able to relay connection and authorization status notifications from the web module to the user interface.	Stian	1:00
S12	The service should be able to relay a request for a Bluetooth device list from the user interface to the database	Stian	1:00
SUM			**79:00**

8.3 Architecture

During this sprint we have kept the overall initial architecture, as described in chapter 5. However, a lot of changes were done to the components themselves, and in some cases also in how they communicate with one another.

8.4 Design and Implementation

8.4.1 User interface layer

During a meeting with the customer we demoed the application, finally with real devices to generate data. The customer was mostly happy with the work done, but had some additional requirements which had not been thought of during the earlier brainstorming sessions on the UI design. Because of this we had to add a few requirements related to UI, despite believing this part of the application being done. Most of these were fairly quick to accomplish, and presented no problem. One however, was a rather big change. The user requested a detailed view of data from the chest unit. This was believed to be a simple task, and we agreed to the last-minute change of requirements. When coding started however, we found this was not as easy to include as we had first believed. When we were getting close to the delivery deadline, we decided to scrap the work done, and inform the customer we were unable to fulfill this part of the application. This is the only thing we were unable to implement which was not due to external factors, and we were not happy about it. There was not much we could do about this however, as time had simply run out. Because of this, we spent more than the allotted hours on UI this sprint, which fortunately did not adversely affect other components. The finished UI can be seen in figure 8.1 and 8.2.

8.4.2 Application logic layer

The work done in the logic layer during this sprint revolved around getting the different parts to talk to each other, specifically making the service talk to each of the other modules in order to perform its function.

Service

Because of the nature of the problem, the service had to run the main event loop of the application in a thread of its own. This turned out to be slightly more challenging that one would think, partially because Android does not offer thread-safety for user-interfaces, and to some extent because the team did not have much experience dealing with threads in Java. The lack of thread-safety for the graphical user interface presented the larger problem, and forced the team to use more workarounds in the data communication between the service and the user interface than what is strictly desirable.

Because Stian was writing the service code from the start of the sprint, and Robin and Dag Øyvind were in possession of Nonin, no external sensors were available to him for testing the service code. Having 2 Nonins would have simplified this

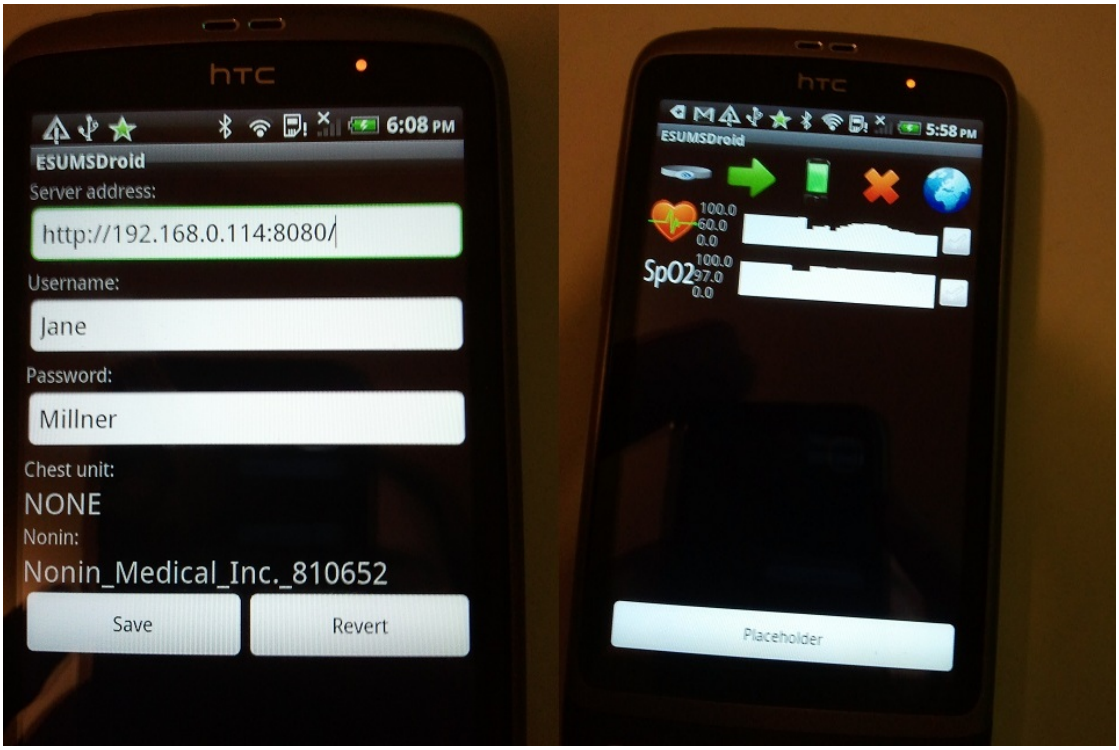

Figure 8.1: Configuration and Main views after Sprint 3.

part of the sprint, but because the one the group had was required for debugging and testing the bluetooth code in particular, a mock-object had to be created to debug and test the data flow between the service and the graphical user interface. The result of this is the FakeDevice class that is a part of the bluetooth package. This turned out to be sufficient for its purposes, and only a very small amount of problems surfaced during the integration phase of the development.

Later in the sprint, it turned out that the most convenient arrangement for finishing the application was to hand over all remaining development tasks to Robin and Dag Øyvind, who were in possession of Nonin, and had written nearly all the code outside of the service, making their experience very valuable for integrating the different parts of the system. During this late stage of the sprint, a lot of the service code was rewritten for various reasons, for example to achieve:

- Properly propagating error reports from subsystems to the user.
- Ensuring that subsystems were being used in a thread-safe way.
- Making sure that tasks were scheduled properly by the eventloop.

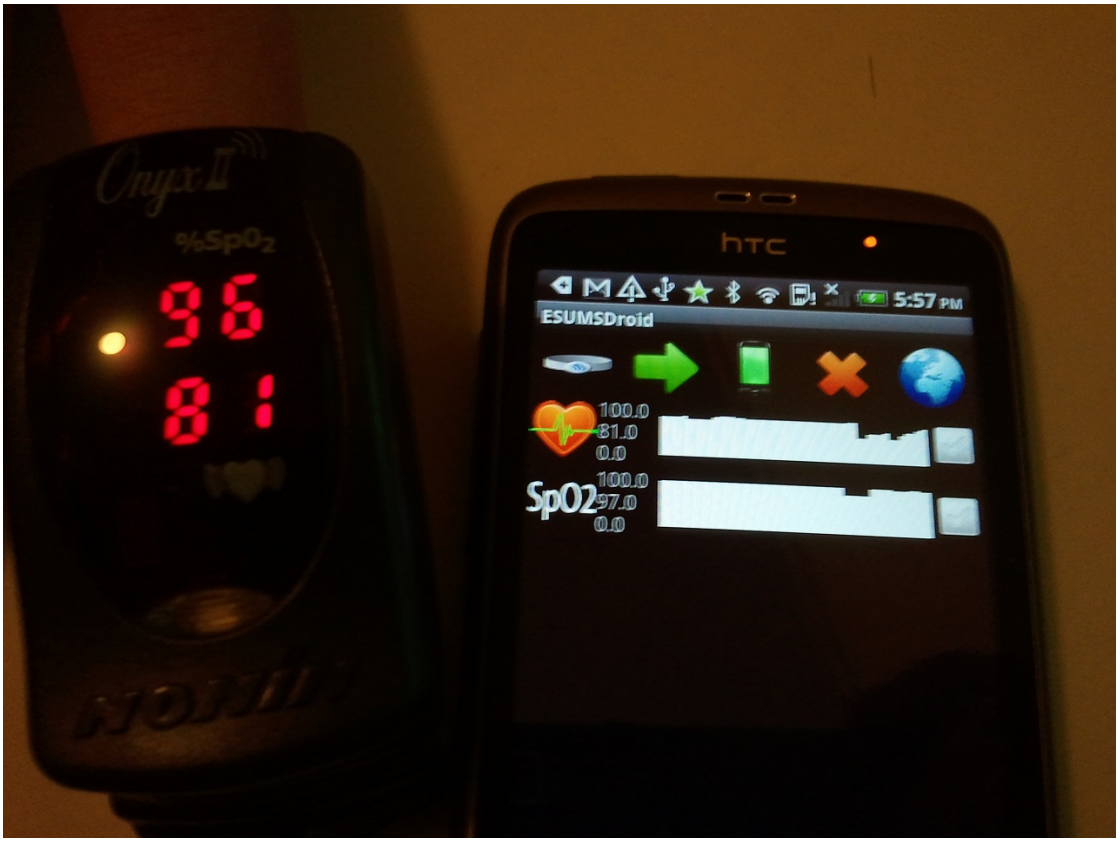

Figure 8.2: Phone running the application getting data from the NONIN sensor.

- Properly initiating devices on startup, and attempting to reconnect when they fail.

In the end, some changes had to be made to accommodate error propagation in particular, that caused other components to be aware of the existence of the service. This situation is not ideal, and was unplanned for. This is because some components had to be redesigned to incorporate threads, which makes it hard to properly report errors in the conventional way of throwing exceptions. By and large, it would have been possible to work around these issues, but with the time remaining of the sprint, it did not seem ideal at the time.

Bluetooth module

Most of the high-level work on the bluetooth module had already been performed at this point, but it turned out to be very difficult to debug bluetooth on android, and we should definitely have done more extensive testing previous to this. Even

so, everything looked to go very smoothly from the start. For example, after receiving the chest unit, hooking up the few missing lines of code and starting the application, the first thing that happened was that data flowed all the way to the graphical user interface, which crashed because it did not know how to handle floating point numbers (Obviously easily fixed). Most of the work after this went into polish, making it easy to define bluetooth devices, more easily accommodating their creation in the service and integrating with the rest of the application. As expected, a few bugs in other parts of the application were uncovered while integrating the chest unit, but nothing that was very time consuming. What proved hard with the bluetooth part of this sprint was the difficulty in debugging, and more specifically in reproducing errors. The abstract device code which had worked perfectly fine on the HTC Desire did not work very well at all on the Samsung Galaxy. Occasionally, the bluetooth socket creation would fail here, for no apparent reason, and other times it would work just fine.

This is probably related to a workaround we used to create bluetooth sockets when we were unable to discover the bluetooth services we wanted to connect to on the peripheral device. We were unable to find the available services on both Nonin and the Chest Unit with a bluetooth service discovery tools[1], and android requires the UUID of the service we are connecting to, in order to establish a connection. This would be a 5-minute fix with the UUID of the services we were using. The code to do so is already in place, and commented out.

Webserver module

While most of the code required for the web service was present at this stage, it was entirely untested. It also turned out that we did not have the identifiers of the data types we were sending to the web service, making it impossible for us to actually correctly send data here. Testing on this started a little late as we were unsure of how to handle the missing data type identifiers. We would have been able to look these up on the server from the handheld, or maybe even create them if they did not exist, but both of these solutions would entail a massive amount of work. For this reason we decided to contact the customer to see if they had the identifiers of the data types, which at the time, we thought would let us use the code we had more or less unchanged. We did get a list of identifiers, but they were obtained fairly late, and testing did not commence until the weekend of 19th to 21st of November. Testing here was troublesome in many aspects. One of the issues was in setting up the web server - this had already been done in sprint 2, but between then and sprint 3, some of the setup had broken. Port forwarding through to the virtual machine that was running the web server proved to be rather

[1] Linux commandline tool sdptool.

troublesome, and somehow it was causing all sorts of other issues (Unrelated to this project) on the host machine as well, making it rather a big time-sink.

When the server was up and running, and was connectible, testing turned out to be a lot of work. This is in very large part because of how difficult it is to decode the tracebacks from ksoap2. The library itself is also not very well documented, and a lot of trial and error was involved in getting things working. Because of how much of the web service code we had already written, we had thought that it would be relatively easy to integrate this into the application, but that turned out to be wrong. In the end, nearly all existing webservice code was rewritten to be simpler. The interface exposed to the application logic layer was changed significantly, and a lot of last-minute changes were made. For example, we decided at the very end of the sprint to run data transmission in a thread of its own, so as to avoid blocking the service while transmitting. This deprecated the sendData method that was only tested and integrated a few days earlier. The interface exposed to the service changed slightly - it now checks whether a data transmission is running, and it will not attempt to send any data if there is a transmission going on currently, and then checks back later on. This is in part because of the actual latency involved in transmitting data. Since the webservice did not support sending many data points in batch mode, we were instead sending each data point one after another, which incurs significant overhead in the form of both bandwidth and latency. The decision to run data transmission in a thread of its own also changed the interface between the service and the webserver module in another way - namely that the thread is now telling the service about the connection status, instead of the service polling it periodically. That means that communication between the two is now two-way, which is not ideal and is a deviation from the initial design. Because it is threaded, the component now looks more like a sub-application than a library for use by the service, as was initially planned. However the threaded part of it is small and proved to be a breeze to test.

8.4.3 Data persistence layer

Nearly no work went into the data persistence layer in sprint 3. Of note, we decided to store the names of icons in the data storage layer, so that no data types are hardcoded into the UI. This facilitates a fully transparent view of data for the UI - every data type can be treated exactly the same, and it will just work. In addition we added a column for data type web service identifiers - the integer ids required by the web services to work with data. The core code of the layer remained exactly the same, the only difference was the addition of two fields to the SQL table for data types. We did discover a few things we were unhappy about with the data persistence layer in this sprint. For example, there is no way of relating two datapoints that naturally belong together, such as for example a

heart rate measurement and the confidence of that measurement.

8.5 Testing and results

Most of the testing that took place in this sprint happened on the HTC Desire[2], connected to Nonin. Some testing was also done in the Android emulator, but this does not support bluetooth, meaning that an actual phone was required for testing the system as a whole, even for smaller debugging tests. The application was also briefly tested against the Chest Unit, and it was tested against a webserver that was installed by the team on a private machine for that purpose only. Unit tests for the datastore were slightly expanded to check a few edge-cases that had not been thought of before, and unit tests for the webservice code were created. By and large, all tests were passed, however the tests were somewhat lacking in some cases. None of the tests that were executed put a significant amount of stress on the application as a whole, for example. A proper stress test would involve being connected to the bluetooth sensors for large stretches of time while simultaneously being disconnected from the internet, and then later connect to the internet again, before disconnecting the bluetooth sensors. This was not done because there simply was not time for it.

While the application was finished, and fulfilled the requirements, there were still some issues with it that team was not entirely satisfied with. As a prototype it does what it is supposed to do, the problem domain has been mapped out and understood well, it has been solved, but not as well as it could have been done. The issues range from lack of optimization, design deviations, bugs that weren't understood well, and a last-minute feature request from the customer that was not satisfied. A full list of the known issues and some suggested fixes and improvements have been provided with the developers guide. The main issue the application suffers from is that it takes a very long time to send data to the webserver while simultaneously reading from a bluetooth sensor, even though this is reasonable fast while not reading from one. The cause of this is currently not well understood.

8.6 Sprint retrospective

This sprint saw a reorganization of the team, into a pure developer group, and a pure writing group. This change was made to allow the team members to maintain focus on a single task. This turned out to be a great idea, but not one without

[2]Because of the bluetooth specific issue we encountered with the Samsung Galaxy, see 8.4.2

problems. We also extended the sprint by two weeks, as the work on application and the report took longer to finish than we had expected.

The single biggest change that happened in this sprint compared to the other ones is that the group decided that with the deadline coming closer, more work needed to go into writing this report. That resulted in Dag Øyvind and Robin more or less being given the task of completing everything related to code that still had to be done, and everyone else starting to work fulltime on the report. This division of responsibilites happened with a little more than a week remaining on the sprint. Since they live in the same house, this turned out to be a reasonable arrangement once integration was being performed, as the devices could be brought there and integration testing could be arranged practically. On the other hand, since the late phase of sprint 3 saw a very large amount of changes to the code, it also resulted in Robin and Dag Øyvind being the only ones who were by now really familiar with all aspects of it. This was inconvenient when it came to writing the report, as the team was reliant on communication with them for some parts of the report to be done, while they were also stressed out about completing the application on time. Some compromises and priorities had to be made for the sake of the report and the application both.

A lot of the work done here should also have been done earlier. Most notably, the webserver responsibility should probably have been delegated to someone else than Thomas while he was ill, so that the development of this component did not stall as much as it did, resulting in almost all testing of this being done at the very end of this sprint. The lesson here is that two people should have been working at this from the start to minimize the impact an illness would have on the development process, as we had predicted that people would get ill.

We can tell from the burndown chart in figure 8.3 that progress was fairly steady throughout the sprint, with the notable exception of the last week. This was due to a large amount of work going in towards the end, as the deadline was quickly approaching. The two developers were quickly finishing tasks, and fixing bugs as they appeared. The burndown chart shows all tasks completed at the end of the sprint, while we have noted elsewhere 9.2 that some issues were unresolved. This means our tasks in the backlog did not completely capture the requirements of the customer. Had we known this earlier we would have changed our process, but as this was at the very end of the project there was precious little we could do. We still believe our process was valid, but this was an obvious hiccup.

Table 8.2: Sprint backlog for sprint 3 with time spent per task

ID	**Task**	**Responsible**	**Time estimate (h:m)**	**Time spent (h:m)**
T1	Set up unit testing framework	Robin	4:00	1:30
T2	Data store tests	Robin	2:00	2:00
BT1	The Bluetooth module should be able to receive a prompt for a search for paired Bluetooth devices in range.	Dag Øyvind	1:00	4:30
BT6	The Bluetooth module should retry a set number of times if no connection is made.	Dag Øyvind	0:30	0:00
W1	The web module should be able to receive prompt to connect to the ESUMS server.	Thomas	0:30	0:00
W2	The web module should be able to connect to the ESUMS server.	Thomas	0:30	3:00
W3	If the web module cannot connect to the server it should retry a set number of times.	Thomas	0:30	1:00
W4	If the web module cannot connect after the set number of times it should notify the user, and offer an option to retry.	Thomas	0:30	0:00
W5	The web module should be able to authorize with the ESUMS server.	Thomas	0:30	0:00
W6	If the web module cannot authorize with the server it should notify the user, and have him contact an administrator (username or password are incorrectly set).	Thomas	0:30	1:00
W7	The web module should be able to send data to the web server, as specified by the WSDL.	Thomas	1:00	1:00
W8	The web module should be able to receive responses which verify the data has been transmitted and notify the service of the success.	Thomas	0:30	0:00

Continued on next page

Table 8.2 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)	Time spent (h:m)
W9	The web module should be able to resend data a set number of times or, if the retries fails, notify the service of the failure.	Thomas	1:00	1:00
UI1	The user interface should be able to select/deselect Live Mode.	Dag Øyvind	3:00	2:00
UI2	The user interface should be able to display recent data from Bluetooth device or, if no data exists, notify the user.	Dag Øyvind	1:00	0:00
UI6	It should be possible to change configuration details via the user interface by entering an admin password	Dag Øyvind	3:00	0:00
UI7	It should be possible to click on the icons for the different statuses in the user interface.	Dag Øyvind	1:00	0:00
UI8	It should be possible to click on each graph to open a fullscreen view of that graph.	Dag Øyvind	3:00	0:00
UI9	The range of the graphs' axes should adjust dynamically to span the data fully	Dag Øyvind	1:00	1:00
UI10	Add thresholds for normal range of the various measurement data.	Dag Øyvind	2:00	1:00
11	The service and the user interface should be able to talk to each other and exchange necessary information and objects.	Stian	15:00	39:30
I2	The service and the Bluetooth module should be able to talk to each other and exchange necessary information and objects.	Robin	15:00	20:00
I3	The service and the storage module should be able to talk to each other and exchange necessary information and objects.	Stian	10:00	15:00

Continued on next page

Table 8.2 – continued from previous page

ID	Task	Responsible	Time estimate (h:m)	Time spent (h:m)
I4	The service and the web module should be able to talk to each other and exchange necessary information and objects.	Robin	10:00	1:00
S4	The service should be able to relay connection and authorization status notifications from the web module to the user interface.	Stian	1:00	0:00
S12	The service should be able to relay a request for a Bluetooth device list from the user interface to the database	Stian	1:00	2:00
SUM			**79:00**	**96:30**

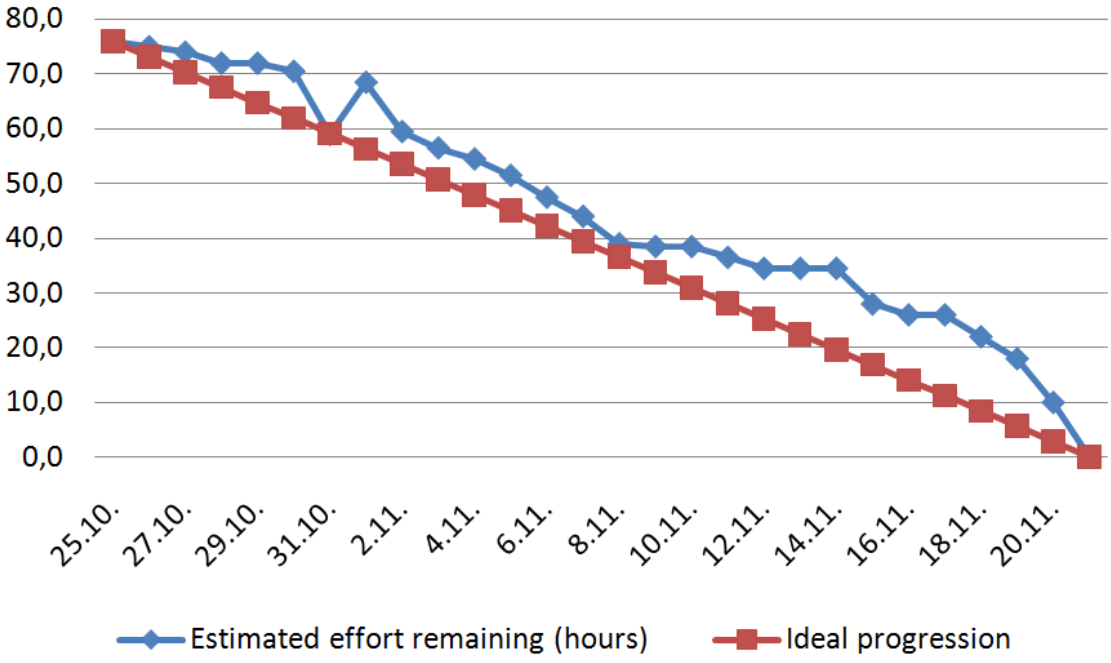

Figure 8.3: Burndown chart for sprint 3

Chapter 9

Project Evaluation

This chapter will contain the evaluation of various aspects of the work done in this project, as well as the course TDT4290 itself.

This chapter will consist of these sections:

- **9.1 Work process:**
  This section discusses various aspects of the work process, including things that worked well and things that should have been done differently.
- **9.2 The product:**
  This section discusses the product that was delivered.
- **9.3 Customer relations:**
  This section details our opinion on how well the collaboration with the customer representatives worked.
- **9.4 Supervisor relations:**
  This section details our opinion on how well the collaboration with the supervisors worked.
- **9.5 Further work:**
  This section discusses possible improvements and extensions that we recommend for ESUMSDroid in the future.
- **9.6 Evaluation of the course:**
  This section contains our views on the quality of the course itself, and we suggest some areas that could be improved in future iterations of the course.
- **9.7 Concluding remarks:**
  This section gives our final comments and a conclusion that tries to sum up our experiences from the project in a few words.

9.1 Work process

9.1.1 Development process

We used Scrum as the development model in this project. The reason for this was mostly it's flexibility as opposed to the more rigid waterfall model. But using Scrum also had it's difficulties. None of the team members had any previous experience using Scrum, so in the beginning, there was a bit of trial and error in getting used to the process. It also took some time deciding how the product backlog should be structured. It can be argued that Scrum was not the best way to approach this project, as we already had a quite firm requirements specification. However, the flexibility of the Scrum model proved to be an asset, as it made it simple to assign tasks to whoever was free and able, and everyone were able to work on varied tasks throughout the project. We did Scrum meetings 2-3 times per week, as planned. Each sprint was initiated by a sprint planning meeting and concluded with a sprint retrospective meeting. We were not able to have a sprint review meeting with the customer at the end of each sprint, but we used the following customer meeting after a sprint ended to demonstrate the application for the customer.

The fact that we were lacking information about the communication protocols used by the chest unit and the ESUMS web server for a very long time, meant that we were not able to implement any functional requirements fully in the first sprint. We had to structure the product backlog into tasks that were divided between the various modules of the application, so that tasks that did not depend directly on the communication modules could be implemented early. This turned out to work well, and we did not have too much trouble "plugging in" the communication modules later.

9.1.2 Work routines

For reasons explained in section 2.6.3, we were not able to get all the team members together in the same room very often. Instead, we had to spend considerable amounts of time working separately, using e-mail and IRC for communication. We feel quite sure that the efficiency of the work would have been better if we had spent more time working in the same room. Unfortunately, this was generally not an option. IRC would have worked better also if people were logged in more often. However, Robin and Dag Øyvind actually live in the same house, so they were able to cooperate quite effectively on their tasks. There were of course positive aspects to working so much separately as well. It allowed a lot of flexibility in choosing when to work. Some people work best in the morning, while others prefer to work in the evening, and such preferences were easy to meet in this project.

Besides schedule problems, we had some communication problems, where misunderstandings occured due to difference of language.

9.1.3 Workload

The time spent vs. time planned per activity is given in table 9.1 and illustrated graphically in figure 9.1. It is immidiately apparent that the estimated relative division of hours per activity was quite accurate, with some noteable exceptions. It seems that very little time was spent on planning. This is most likely due to quite a bit of planning work being logged as project management, which also helps explain the relatively high percentage of project management. It is also obvious that we seriously overestimated the amount of time needed for project evaluation, while we underestimated the amount of time needed for report writing. It should be noted also here that some of the work which was logged as report writing probably should have been logged as evaluation. Finally, the amount of time spent on sprint 3 is much higher than sprint 2. The only explanation for this is that too little time was spent on sprint 2 (in large part due to many team members falling ill in that period), and quite a bit of work had to be transferred to sprint 3. The amount of work required in sprint 3 to finish the application also turned out to be higher than expected, which forced us to extend sprint 3 into the two week period that was originally set off for report finalization and presentation. This led to a quite a lot of stress towards the end of the project. This can clearly be seen in figure 9.3. Notice the very large increase in activity in the last week of the project. This goes to show that a deadline can be a very effective motivator, but in retrospect, the project would have been a more pleasant experience if the distribution of workload over time had been more uniform. The variations in the curves throughout the project are due to many factors, among them sickness and periods with lots of activity in other courses.

Looking at the total time spent working on the project, it is very apparent that we spent less time than planned. The simplest way to explain this is that the motivation for and prioritization of this project as compared with other courses and activities (such as work) were different for the various team members, and some team members spent far less time on the project than the alotted 24 hours per week, as can be seen in figure 9.2. This figure also shows that the total workload was divided quite unevenly between the team members. This again is due to varying motivation for and prioritization of the project. However, everyone has contributed greatly to what was delivered in the end, and most of the time, we functioned well as a team.

Table 9.1: Time spent vs. time planned per activity

Activity	**Planned time (hours)**	**Planned time (percentage)**	**Spent time (hours)**	**Spent time (percentage of total)**	**Spent time (percentage of planned)**
Project mgmt.	265:30	15.0 %	256:08	17.6 %	96.5 %
Study	354:00	20.0 %	299:43	20.6 %	84.7 %
Planning	88:30	5.0 %	17:53	1.2 %	20.2 %
Req. spec.	88:30	5.0 %	68:00	4.7 %	76.8 %
Sprint 1	88:30	5.0 %	65:10	4.5 %	73.6 %
Sprint 2	221:15	12.5 %	104:10	7.1 %	47.1 %
Sprint 3	221:15	12.5 %	242:55	16.7 %	109.8 %
Evaluation	88:30	5.0 %	16:00	1.1 %	18.1 %
Report writing	265:30	15.0 %	363:08	24.9 %	136.8 %
Presentation	88:30	5.0 %	24:30	1.7 %	27.7 %
TOTAL	**1770:00**	**100,0 %**	**1457:37**	**100.0 %**	**82.4 %**

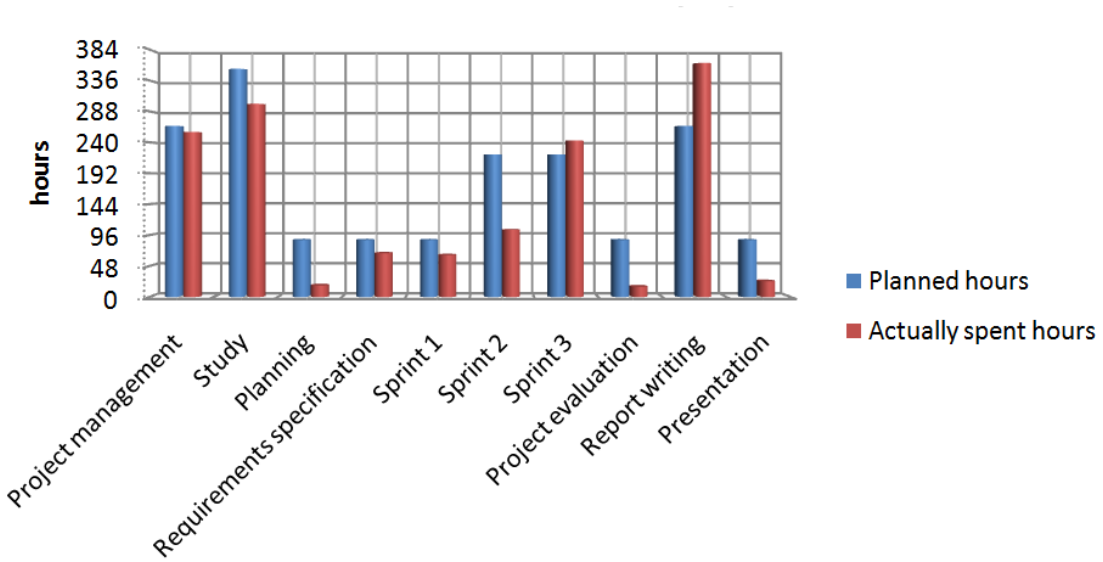

Figure 9.1: Time spent per activity.

Weeks/Team members	Øystein	Dag Øyvind	Stian	Robin	Yushan	Thomas	Total (all team members	Average (all team members)	Standard deviation (all team members)
35	8:50	3:30	9:30	8:55	9:40	8:15	48:40	8:06	2:18
36	16:30	12:00	19:35	12:55	18:58	14:30	94:28	15:44	3:08
37	27:02	13:30	22:00	14:40	25:25	19:00	121:37	20:16	5:33
38	27:22	16:00	20:00	1:30	18:15	20:10	103:17	17:12	8:35
39	19:35	10:45	20:00	18:15	17:10	15:00	100:45	16:47	3:27
40	31:20	19:30	25:00	17:40	16:55	16:40	127:05	21:10	5:51
41	18:00	16:00	16:00	14:40	14:25	16:30	95:35	15:55	1:18
42	20:25	7:00	20:00	8:30	15:40	2:00	73:35	12:15	7:33
43	20:00	12:00	14:00	9:00	15:30	11:30	82:00	13:40	3:48
44	19:20	13:00	34:00	20:15	20:00	20:30	127:05	21:10	6:53
45	22:10	23:00	21:00	14:30	21:15	16:30	118:25	19:44	3:24
46	50:25	34:30	43:00	36:00	24:35	40:30	229:00	38:10	8:44
47	26:20	26:00	29:30	17:00	17:45	19:30	136:05	22:40	5:14
Total (all weeks)	307:19	206:45	293:35	193:50	235:33	220:35	1457:37	242:56	46:52
Average (all weeks)	25:00	16:49	23:53	15:46	19:10	17:57	118:38	19:46	3:48
Standard deviation (all weeks)	6:04	5:24	6:13	5:25	4:03	5:31	24:28	4:04	

Figure 9.2: Time spent per week per team member

9.1.4 Documentation of project work

In the beginning of the project, we used an online hour logging system called Yast [11] to keep track of the time spent on various activities. After a few weeks, however, we realized that this system was not flexible enough, and we created a Google spreadsheet to replace Yast. The Google spreadsheet was later expanded to manage the sprint backlogs as well. All in all, it worked really well. It made it easy to keep track of progress and to compare the work effort put into the project by all the team members. Also, logging of hours was quick and simple.

9.2 The product

We are pleased with the product that we can now deliver to the customer. It is a working prototype that successfully receives measurements from two different Bluetooth units, is able to display measurements to the patient through the GUI, and is able to send the data to the ESUMS web server, where it can be accessed by health personnel. The application fulfills almost all the requirements set by the customer, and considering our limited experience with Android from the start, as well as the complexity involved in creating an application to be used in a medical setting, we are happy with this result. Of course, there is room for extended functionality, and there are some details that we are not 100 % happy with, but

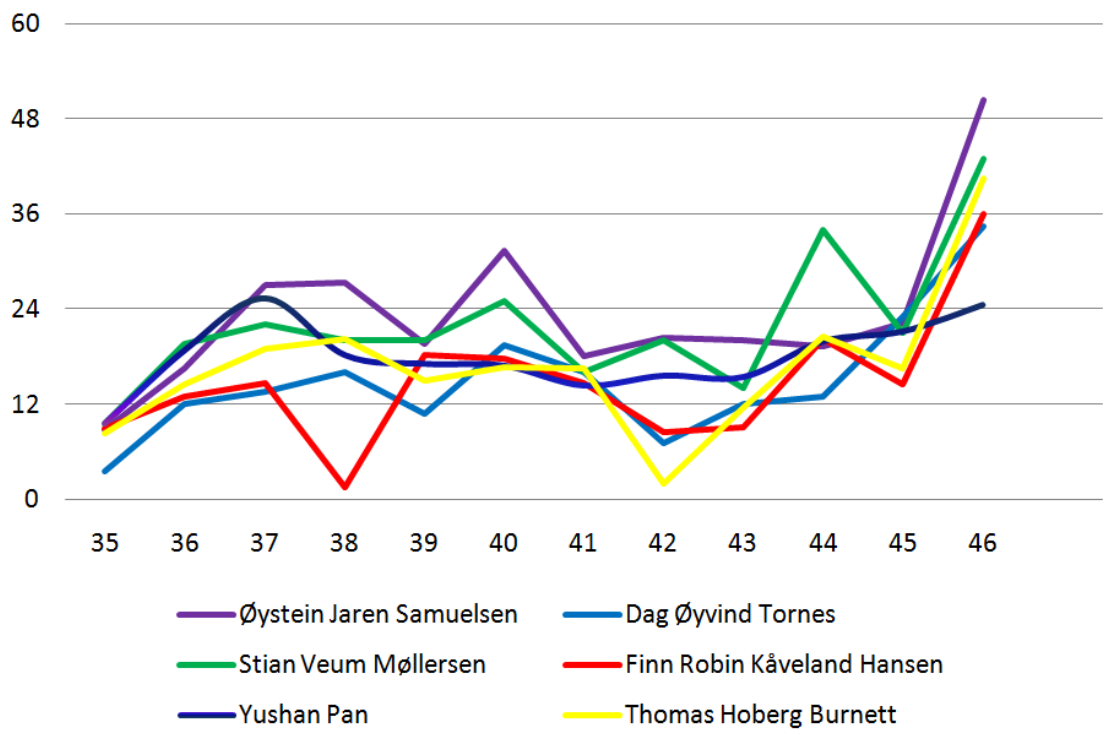

Figure 9.3: Time spent per person per week. Week 47 is not included, as that week only contains three work days.

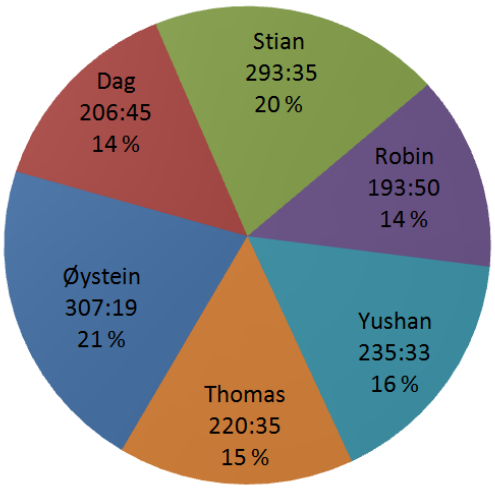

Figure 9.4: Total time spent per person throughout the entire project

in the end we are happy with what we have accomplished.

The issues we were unable to resolve have been thoroughly documented for the customer (see appendix B), and we believe they should all be surmountable without too much work.

9.3 Customer relations

We were lucky to have customer representatives who understand the technical aspects of developing a software system. This allowed us to communicate with the customers on the same technical level as within the team, and the customer was always able to give advice that was valuable to us. However, we would have liked to meet with the customer representatives a little more frequently than just once every other week. The response time on e-mail was also occasionally a little high, which led to some periods where parts of the development had to be put on hold for a few days until an answer was received. On a positive note, though, they were nearly always able to provide the answers we needed. All in all, we had a good dialog with the customer representatives, given the limited amount of time they had available to work with us.

9.4 Supervisor relations

We had a meeting with the supervisors every week throughout the project (with a few exceptions) and for these meetings, we normally followed the routines set in section 2.6.2. The supervisors often encouraged us to spend more time on the project, as the number of hours per week was consistently a little low. That turned out to be good advice, because we ended up with quite a lot of work left towards the end of the project, which led to a quite stressful situation. The supervisors also provided valuable feedback on the work process and the progress of the report writing. The entire team did have problems understanding Sundar's English, as he speaks with a quite difficult accent. Especially Yushan had problems with this, which is not that surprising, seeing as he did not have much experience with English himself. We did manage it with a bit of patience, though. The problem would have been less severe if Sobah had been able to attend more meetings, as her English was far more comprehensible. Unfortunately, she was unavailable for most of the advisory meetings. In any case, communication problems aside, we had a good dialog with the supervisors, and their continuous feedback on our work was invaluable for the final result.

9.5 Further work

There are quite a few things that can be done to improve and extend ESUMSDroid in the future. This section lists some of them. For a more comprehensive list, see appendix B.5.

9.5.1 Implement normal mode

Originally, the intention was for us to implement both normal mode and live mode in the handheld client. Live mode means that the chest unit sends data to the handheld in a continuous stream. Normal mode is more relaxed, sending bunches of accumulated measurements periodically. Normal mode requires that the data points sent from the chest unit contain time stamps. The chest unit does not currently support time stamps, so it was impossible for us to implement normal mode. The lack of time stamps also means that any measurements done by the chest unit when ouside the range of the handheld will be lost. We were told that this is a planned feature, and we made sure the code of the handheld client is ready for it, so it will be relatively painfree to implement.

9.5.2 Fully support external sensors as plugins

Currently, code modification is required to incorporate new sensors into the code, this includes modifying the application logic layer slightly, which is not desirable. We wanted to use Java reflection to ensure that adding a new class to the bluetooth package would be enough, but did not find time for it.

9.5.3 Add more external Bluetooth sensors

The Bluetooth communication module was implemented with extensibility in mind. Our solution allows for fairly quick and simple implementation of additional Bluetooth sensors. One such device which has been mentioned in the documentation we received from the customer is a Bluetooth enabled personal weight.

9.5.4 Graphical detail view of data

This was a late feature request from the customer, where the idea is to have a view in the graphical user interface that will show more detail about data, both what is being logged, and what kind of data that is. It was never fully specified, but it should at least support seeing a graph in fullscreen, as well as have some indication of whether the data currently coming in is in a “good” range.

9.5.5 Moving more configuration options out of code

Currently, a lot of configuration is being done on the class-level, that is to say that classes have static constants, where they could instead be using the configuration package to make the application more dynamic. We did not take the time to start moving these constants out of code and into configuration, but we feel that it would make the application a better product to do so.

9.5.6 Use data type information from ESUMS Server

The ESUMS webserver has a lot of information about data types, as well as data ranges in general, and for specific patients that the handheld application currently does not look up, or use at all. While the information is not very complex to look up from the application code, it would be a relatively large amount of work to incorporate it into the rest of the application code, so we did not prioritise this. Currently the data type specific boundaries and ranges are instead hardcoded into the application.

9.6 Evaluation of the course

In this section, we give our opinion of the course TDT4290, and we suggest some possibilities for improvement for next year.

- NTNU provided us with access to an SVN repository on 22 September. This is far too late. We had already set up a different version control system, and we had started using it.

- The workload for the course given in the information booklet is too high. It gives a total workload of 312 hours per student for the whole project, based on the fact that 24 hours should be spent per week on a 15 credit course. However, the project lasts for 12,29 weeks, and 312 hours divided by 12,29 weeks equals about 25,4 hours per week. We have worked with 24 hours as the norm, which we feel is the maximum amount of time that can be spent, considering most students take two additional subjects in parallell with TDT4290.

- The Friday seminars were very interesting, and we got lots of useful tips especially from the seminars on technical writing and presentation technique.

9.7 Concluding remarks

It is likely that we will come to look back at this project in the future as one of the most educational as well as stressful experiences in the course of our studies. Being put in a team of mostly strangers and having to solve a large problem together was definitely interesting, and even though we have all done team projects in past, it has never been on such a large scale, and never in such a realistic setting. Having an actual customer to report to was certainly an important motivator, and for certain also an invaluable experience, relevant for an actual work setting.

We are happy with the product we delivered, and even though there is room for improvement in the software, it is a fully working prototype of an ESUMS handheld client on Android, which is what our customer wanted from this development effort.

References

[1] Ingrid Svagård, Ståle Walderhaug, Hanne Opsahl Austad, Anders Kofod-Petersen, and Erlend Stav. *ESUMS Deliverable 2: ESUMS System specification - Handheld and Server specifications*. SINTEF, January 2010.

[2] Mountain goat software: An introduction to scrum, 2010 (accessed 22 Nov). `http://www.mountaingoatsoftware.com/system/presentation/file/30/RedistributableIntroToScrum.ppt?1267636399`.

[3] Google. Current android version distribution, 2010 (Accessed 2 Sep). `http://chart.apis.google.com/chart?&cht=p&chs=460x250&chd=t:7.9,15.0,0.1,40.8,36.2&chl=Android1.5|Android1.6|Other*|Android2.1|Android2.2&chco=c4df9b,6fad0c`.

[4] Ingrid Svagård, Frode Strisland, Jon Vedum, Trine Seeberg, and Morten Borch. *Enhanced Sustained Use Monitoring System: Deliverable 1: Requirements Specification*. SINTEF ICT, September 2009.

[5] SINTEF. Sintef website [online], 2010 (accessed 28 Sep). `http://www.sintef.no/`.

[6] Wikipedia. Wikipedia entry for scrum [online], 2010 (accessed 25 Sep). `http://en.wikipedia.org/wiki/Scrum_(development)`.

[7] Mountain Goat Software. Mountain goat software: User stories, 2010 (accessed 26 Oct. `http://www.mountaingoatsoftware.com/topics/user-stories`.

[8] Bitbucket. Bitbucket [online], 2010 (accessed 3 Sep). `http://bitbucket.org/`.

[9] Mercurial. Mercurial [online], 2010 (accessed 9 Sep). `http://mercurial.selenic.com/`.

[10] DropBox. Dropbox website [online], 2010 (accessed 18 Sep). `http://www.dropbox.com/`.

[11] Yast. Yast [online], 2010 (accessed 9 Sep). http://yast.com/.

[12] Google. Google documents [online], 2010 (accessed 1 Sep). http://docs.google.com/.

[13] Sintef. Freempower, 2010 (Accessed 22 Sep). http://www.sintef.no/Projectweb/MPOWER/.

[14] W3. Web services description language (wsdl) 2.0, 2010 (accessed 15 Nov). http://www.w3.org/TR/wsdl20/.

[15] W3. Soap, 2010 (accessed 03 Oct). http://www.w3.org/TR/soap/.

[16] W3. Soap intro, 2010 (Accessed 19 Nov). http://www.w3.org/TR/2007/REC-soap12-part2-20070427/#intro.

[17] Stefan Haustein and James Seigel. ksoap2, 2010 (accessed 06 Oct). http://www.ksoap2.sourceforge.net/.

[18] Wikipedia. Android wikipedia entry, 2010 (Accessed 2 Sep). http://en.wikipedia.org/wiki/Android_(operating_system).

[19] Google. About the android open source project, 2010 (Accessed 2 Sep). http://source.android.com/about/index.html).

[20] Mercury News .com. Google's android software dominates u.s. smartphone market, 2010 (Accessed 2 Sep). http://www.mercurynews.com/top-stories/ci_16493024?nclick_check=1).

[21] Google. Activity, 2010 (Accessed 2 Sep). http://developer.android.com/reference/android/app/Activity.html.

[22] Google. View, 2010 (Accessed 2 Sep). http://developer.android.com/reference/android/view/View.html.

[23] Google. Activity lifecyle, 2010 (Accessed 2 Sep). http://developer.android.com/guide/topics/fundamentals.html#actlife.

[24] Google. Service, 2010 (Accessed 2 Sep). http://developer.android.com/reference/android/app/Service.html.

[25] Google. Service lifecyle, 2010 (Accessed 2 Sep). http://developer.android.com/guide/topics/fundamentals.html#servlife.

[26] Google. Platform versions, 2010 (Accessed 2 Sep). http://developer.android.com/resources/dashboard/platform-versions.html.

[27] Google. Android sdk, 2010 (Accessed 24 Sep). http://developer.android.com/sdk/index.html.

[28] Google. Android development tools plugin [online], 2010 (accessed 12 Sep). http://developer.android.com/sdk/eclipse-adt.html.

[29] Checkstyle. Checkstyle 3.5 [online], 2010 (accessed 13 Sep). http://checkstyle.sourceforge.net/.

[30] Wikipedia. Rs-232 wikipedia entry, 2010 (Accessed 29 Sep). http://en.wikipedia.org/wiki/RS-232.

[31] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison Wesley, second edition, 2003.

[32] Wikipedia. Multi-tier architecture wikipedia entry, 2010 (Accessed 2 Oct). http://en.wikipedia.org/wiki/Three_layerya.

Appendix A

Project plan

A.1 Meeting agenda

Agenda for advisory meeting

TDT4290 - group 9

October 18, 2010

1 Meeting information

Date: Tuesday, 19.10.2010
Time: 08.45 - 09.45
Location: ELROM-G026
Invited: Sobah Abbas Petersen, Sundar Gopalakrishnan, team

2 Agenda

1. Approval of agenda
2. Approval of minutes from supervisor meeting week 41, Tuesday, 12.10.2010
3. Approval of status report for week 41
4. Review and approval of attached phase documents (Final report)
5. Detailed documentation/readme for the application and source code, made specifically for customer use in the future. Should this be part of the final delivery for evaluation?
6. Other issues

A.2 Meeting minutes

Minutes from team - customer meeting

TDT4290 - Group 9

October 25, 2010

Present: Anders, team

1 Improvements and additions

- UI looks good, but create functionality for viewing one graph at a time in more detail.
- Make the graphs able to adjust the scale of the axes to the data.
- Add thresholds for normal range of the various measurement data.
- Add some functionality to monitor Glassfish (whether it actually receives the data we send)

2 Q&A

- Chest unit communication protocol and normal mode?
  - Concentrate on live mode, as there are no time stamps yet. Make placeholder for normal mode.
- Sending data to the server in batches?
  - Should be able to send data in batches to the server. (Previously only individual data points were supported, but it should be possible now.)
- Any more external BT sensors (besides chest unit and NONIN)?
  - Only chest unit and NONIN confirmed, but make sure it's easy to implement new ones later.
- Code conventions don't allow use of short variable names (i) in loops?
  - It's ok to use short variable names in loops (like i), but use descriptive variable names wherever it makes sense to do so.
- Manner of delivery?

  - Anders will look into how we should deliver the code. Probably with subversion. Also developer's manual, user's manual, documentation. Include rationale for choices in the documentation (decisions must be traceable).

3 Borrowing equipment

- Decide when we want to borrow the chest unit. We can borrow it for about a week, and then for the presentation as well.
- Can borrow the NONIN sensor any time. Anders will get it to us within a few days.

4 Other issues

- Can Bluetooth handle multiple simultaneous connections? (It should be able to, but we need to test this.)
- Write up a status document including all the requirements from the original ESUMS requirements spec. and whether they are/will be fulfilled.
- Remember to have a backup plan for demonstration (video).

A.3 Weekly status report

Status report, week 40

TDT4290, Group 9

November 22, 2010

1 Summary

This has been the second week of Sprint 1, which has wrapped up the first phase of implementation. Good progress has been made on the final report.

2 Work accomplished

2.1 Status of documents

- Project plan: Nearing completion, missing a few subchapters (organization, guidelines,).
- Final report: Making good progress.
- Requirements specification: Functional requirements done. Non-functional requirements need some refinement.
- Sprint 1 document: Need to copy things from googledocs to document and sum up the sprint

2.2 Meetings

- 04.10.2010: Team-customer meeting.
- 07.10.2010: Predelivery presentation and advisory meeting.
- 08.10.2010: Sprint evaluation meeting.

2.3 Other activities

- Finished sprint 1, with review and retrospective.
  - GUI requirements from Sprint 1: Missing configuration dialogue.
  - Database requirements from Sprint 1: Done
  - Service requirements from Sprint 1: Missing some control structuring.

2.4 Hours spent

Hours spent per person and category is given in figure 1.

Hours per person per category

	Øystein Jaren Samuelsen	Dag Øyvind Tornes	Stian Veum Møllersen	Finn Robin Kåveland Hansen	Yushan Pan	Thomas Hoberg Burnett	SUM	Percentag
PROM	4:20:00	4:30:00	4:30:00	5:40:00	7:55:00	6:40:00	**33:35:00**	26%
STUD	3:30:00	2:00:00	0:00:00	3:30:00	0:00:00	0:00:00	**9:00:00**	7%
PLAN	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	**0:00:00**	0%
REQ	0:00:00	0:00:00	12:00:00	0:00:00	0:00:00	9:00:00	**21:00:00**	17%
SPRINT1	2:30:00	6:00:00	8:30:00	6:45:00	9:00:00	0:00:00	**32:45:00**	26%
SPRINT2	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	**0:00:00**	0%
SPRINT3	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	**0:00:00**	0%
EVAL	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	**0:00:00**	0%
REPORT	21:00:00	7:00:00	0:00:00	1:45:00	0:00:00	1:00:00	**30:45:00**	24%
PRES	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	**0:00:00**	0%
SUM	**31:20:00**	**19:30:00**	**25:00:00**	**17:40:00**	**16:55:00**	**16:40:00**	**127:05:00**	100%

Figure 1: Hours spent per person and category

3 Problems

- Chest unit is currently being redesigned. Communication protocol uncertain.
- Some issues with the repository (Resolved).

4 Planned work for next week

4.1 Meetings

- Monday 11.10 (0815): Sprint planning meeting
- Tuesday 12.10 (0845): Meeting with supervisors
- "Daily" scrums: Tuesday (0815) and Thursday (1015)

4.2 Activites

- Plan Sprint 2
- Hand in predelivery
- Start working on Sprint 2

5 Other issues?

A.4 Non-disclosure agreement

SINTEF 1

Avtale om taushetsplikt

mellom

Stiftelsen SINTEF

og

.....

Stiftelsen SINTEF er en selvstendig juridisk enhet
og er registrert i foretaksregisteret med organisasjonsnummer 948 007 029

All informasjon partene har mottatt fra hverandre enten skriftlig, muntlig eller på annen måte, skal behandles fortrolig og ikke benyttes uten skriftlig avtale, med mindre mottakeren kan dokumentere at informasjonen

i) var allment tilgjengelig da den ble mottatt
ii) var kjent av mottaker ved mottakelsen
iii) ble mottatt lovlig fra tredjeperson uten avtale om hemmelighold
iv) ble utviklet av mottaker uavhengig av mottatt informasjon.

Partene skal ved forhandlinger søke å løse konflikter som måtte oppstå i forbindelse med eller som er et resultat av denne avtalen. Dersom dette ikke lar seg gjøre, velger partene Trondheim Tingrett som verneting.

For

_____

Dato:

For **Stiftelsen SINTEF**

_____

Dato:

Appendix B

Design and Implementation

B.1 Introduction

This document explores the design and implementation of the ESUMSDroid application on a more detailed level than could be done in the report. Some detail about internal implementation, interfaces between the moving parts of the application, and decisions and trade-offs that have been done during development are documented here. This document assumes a technical familiarity with some of the technologies that have been used for the development of ESUMSDroid, including Java, SQL, constructs such as threading, classes and so on. The document is mainly intended as a useful companion guide for people who wish to fully understand the problems and solutions that were encountered during development. Many of the design decisions that were made are justified here, but this document is not intended to be a reference to the code, Javadoc will be used for this.

B.2 Persistence Layer

As implied by its name, the persistence layer has to store all data that has to be persistent between runs of the application. Essentially, all data logged from a device has to be on the handheld, until it has been received by the webserver. Additionally, configuration settings have to be stored somewhere, and settings that should be changed during one run should persist for the next one. A very simple configuration file class exists for that purpose, but the main portion of the persistence layer is the datastore code.

B.2.1 Datastore

The datastore is the component that offers temporary storage of data on the handheld, until it can be sent to the server. It needs to be able to recieve data from various external sensors, such as Nonin, the ESUMS Chest Unit, and possibly others in the future, hold onto it until it is marked as sent, and then delete it to conserve space.

This maps onto a very typical CRUD scenario. Crucial to the datastore is that it needs to persist even should the application be closed or the phone be turned off. This left us with two relevant ways of implementing data storage - we could use the file system offered by Android, or we could use the database it offers.

While the file system is conceptually the easiest way to do this, it is unsatisfactory in some ways. Using flat files of unstructered data does not naturally lend itself towards flexibility, so adding different types of data at a later date could potentially be difficult. It's also inefficient to search in these files, so we would have to implement some sort of ordering. Deleting from the start of a file requires one to move everything after that in the file, and splicing the start and the end of a file after taking out the middle leaves a similar problem.

It seems likely that had we taken this approach, we would have needed to use some well-known format such as XML or JSON for these files, which would let us rely on already existing, efficient parsers for these formats. However once we've implemented ordering, to give efficient searching, there are still challenges to overcome. For instance, we would require thread safety, relying on file system locks and other safety mechanisms. We need to take a lot of the files into memory to delete from anywhere but the end. Extending the files to include new datatypes could be non-trivial as well, depending on the design chosen to store them.

Once all of these things are taken into account, it starts to become clear that what we really needed was a database system. Android offers SQLite, which offers persistency, thread-safety, and efficient CRUD operations. Not only does this leave the team with less work in implementing the datastore, it is also a well-tested and known framework. Simply put, SQLite takes care of all the requirements the team has established that we need for the datastore, and it seemed the obvious way was to design the datastore around it.

To make this choice as transparent as possible to the rest of the application, it was early on decided that the datastore does not reveal any SQL details to other parts of the application. This means that the interface from the datastore talks in terms of builtin java sequences instead of rows, of classes instead of tables and methods instead of queries. While we never anticipated that the datastore would be changed to use something other than SQLite, it should be a relatively simple task to switch to an equivalent system that uses a different backend.

There were four different, obvious ways to design the database. The first and

most obvious way of doing it was with one big table, with a column for each kind of different data, and the meta-data attached to it, such as a timestamp. The second was to use one big table with a column for datatype, a column for actual data and columns for metadata. Both of these designs were found to be inefficient and less than optimal. The first way has an obvious problem with extensibility. As stated in the quality attribute requirements, it should be easy to modify the application to include more sensors and datatypes in the future. We believe that adding columns to tables that are used extensively in the application could be a frustrating exercise. The second one, while better, is still not very flexible. It also takes up more space than strictly required. For example the name of a data source, or a type, would be saved many times while in reality it only has to be saved once and then referred to. Once we start including descriptions of these things in the datastore, the excess space taken up could be noticable. A third design is to use one table for each sensor. This also suffers from lack of extensibility.

We therefore settled on the fourth design. The database is split into three different tables, corresponding to three different classes in the code. These are DataSource, DataType and DataPoint. A DataSource object corresponds to one row in the DataSource table, and contains only a name and a description. The ESUMS Chest Unit would be an example of something that is considered a DataSource. Adding a new Source to the application at a later date would then not need any modifications to the datastore package itself, it should only require the insertion of a row into the DataStore table. This is also transparent to the code - nothing outside the datastore sees that it is talking to SQL.

A DataType represents some kind of data that is logged by a DataSource. It stores a reference to the DataSource from which it can be logged, it stores a name and a description. It also stores the id that the MPOWER webserver associates with this datatype. Should some DataSource be upgraded in the future, creating a new DataType from it should allow it to log data without modification to any code.

Finally, the table and class we expect to be used the most is DataPoint. All data logged by all sources go into this table. A DataPoint has a DataType attached to it, and therefore indirectly also a DataSource. In addition it stores a timestamp for when it was recieved by the handheld, and a status that allows the datastore to decide whether or not to delete it. Periodically, the application will tell the datastore to wipe all DataPoints that are known to have been recieved by the server.

The primary services the datastore offers to the rest of the application are:

- Persistent storage of a DataPoint.
- Retrieval of all DataPoints that have not yet been dispatched to the server.

- Registering DataPoints as sent.
- Deletion of all DataPoints older than a certain age.
- Deletion of all DataPoints that have been dispatched to the server.

All values returned or used as parameters are either native Java datatypes, or instances of either DataSource, DataType or DataPoint. The classes all allow for inspection of their fields, and offer methods for updating certain fields (Such as the status for a DataPoint, for example).

Here are some example instance that correspond to database rows.

```
DataSource ESUMS = {
  long id = 1;
  String name = "ESUMS Chest Unit";
  String description = "The ESUMS Chest Unit logs a variety of data.";
};
DataType HBPM = {
  long id = 1;
  String name = "HBPM";
  String description = "Heart Beats Per Minute";
  DataSource source = ESUMS;
  long web_id = 5;
};
DataPoint sample = {
  long id = 2713;
  String data = "78";
  long timestamp = 1286392580;
  DataType type = HBPM;
  Status status = SENT;
};
```

B.3 Application logic layer

This layer is where the magic happens, so to speak. It is the brain of our system and where all the functionality is implemented. It consists of three important parts:

- **Bluetooth Module**
  Where all bluetooth communication is implemented and that handles all parsing of incoming data.

- **Service**
  The control class that is responsible for handling tasks and facilitating the viewing of the data from the user interface when needed.

- **Web services module**
  Where all the ESUMS Web Services server communication is implemented and the parsing of internal data to send is done.

Below follows a description of the implementation details of each part.

B.3.1 Bluetooth module

The bluetooth module as a package supports quite a lot of functionality, some of which is currently not used by the application. The main idea here is that there is a mapping between peripheral devices and Java classes, where the classes have a uniform interface so the service can easily use them. For the service, the way to use the Nonin device should not differ from the way it can use the Chest Unit device. This problem naturally lends itself to a solution that uses an abstract base class, which was named AbstractDevice. This is extended to add devices as needed, overriding methods as necessary, and implementing methods that are known to be device-specific, such as parsing of byte-arrays that are read from the underlying bluetooth connection. Devices also need to provide some information about what they do to the data persistence layer, so that information about data points can be sent around to the rest of the application transparently. This happens in the constructor of the device, where a DataStore method named install is called on any DataSource and DataType objects that are needed to create DataPoints internally in the device. On a first run of the application, this means that the objects will be stored to the database, on subsequent runs, they will be fetched, and necessary fields will be put in memory.

Once an AbstractDevice is created, it can be connected to the peripheral device it represents by calling its run method. This is intended to be run in a thread, as bluetooth connections take a very long time to time if they fail to connect, and if the connect method was to block, it could stall many other ongoing operations for as much as 12 seconds at a time. The service creates the thread and starts it when it wants to connect devices. When a connection is made, a flag is set in AbstractDevice, that enables the service to start polling it for data. The rest of the life-cycle of the device happens in the thread belonging to the service, and the thread that was used for connecting the device will die.

The entrypoint for a connected device is its readAndParse method. This method checks whether the underlying bluetooth socket has enough data in it to read from it, and if so, it will naively read into a byte array, that is then passed

into the abstract parse method. The parse method is device-specific, will take a byte array, and create DataPoints from it if it can. If the incoming data represented something that is not interesting to store on the handheld, for example battery state, an empty list of DataPoints may be returned. This is normal, and happens with both Nonin and the Chest Unit implementation. If the parse method returns any DataPoints, they will be added to an internal buffer in the device class, and the service will fetch these points periodically, causing the buffer to be reset. The service takes care of storing data created here, so a device is only concerned with reporting incoming data to the outside, and takes no further action on data once it has been created. For a guide on how to use the bluetooth package to add peripheral devices, AbstractDevice.java and Nonin.java should provide adequate example code. Currently, Java reflection is not being used to automatically incorporate new subclasses of AbstractDevice into the service, so these have to be added manually to the classes that should be loaded, in AbstractDevice.java. This is something we would have liked to improve, but did not find time for it.

Of note, there is a class in the package that is not currently in use by the rest of the application. BluetoothScanner will obtain a list of bonded devices from Android, and can then initiate a bluetooth scan to check which of these that are discoverable. The intention here was that we could periodically scan our surroundings, and attempt to connect only when we knew devices were nearby. This approach was abandoned at an early stage, but the concept seemed useful, so we let the class stay in case it could be used at a future date.

B.3.2 Service

, The service is implemented as a subclass of the Android class *Android.app.Service* and is started with the parameter *START_STICKY*. The *START_STICKY* parameter means that the service is handled by the operating system as an important process and will not be killed unless explicitly killed by the program itself and will be scheduled to restart if it crashes. This ensures that the uptime will be as good as the Android operating system can provide.

The service itself is implemented as its own thread which calls the methods provided by the other classes in the system, thereby performing the tasks specified by the requirements in chapter 4. Whenever an event is triggered in the thread it will pass a Message to a handler which will in turn call the appropriate method in the receiving module. The reason for doing it this way is to avoid concurrency problems that sometimes present themselves when operating with multi-threaded systems [1]. One cycle in the service consists of several small tasks:

[1] Android is very strict when it comes to altering views outside of the main activity thread, more about that in section B.4.1

- **Fetch data:**
  Poll a bluetooth device for new data.
- **Store data:**
  Store the new data in the database.
- **Check the chest unit battery**:
  Checks the chest units battery status [2]
- **Display data:**
  Send the new data to the messenger for transport to the user interface.
- **Send data:**
  Send unsent (see section B.2.1 for details) data to the web services module.

In figure B.1 you can see the service cycle as a sequence diagram.

To communicate with the user interface we have implemented a subclass of *Android.os.Binder* called ActivityServiceMessenger. This class will track the state of the service and the user interface and provide methods for communication between the user interface and the service. It is implemented in such a way that both the service and the user interface can blindly pass objects to it without having to worry about them reaching its destination, which greatly reduces complexity within the program.

The main eventloop of the service that does everything described above, takes place in its run method. While a lot of things happen here, the method is not nearly as complex as it was at earlier stages of development. Most of the items that happen here are delegated to other methods, to facilitate both easier debugging, and easier reading of the code. The service continually tries to reconnect failed devices. There were some other possible solutions discussed to solve the problem of disconnected devices, and this is the reason for the presence of the BluetoothScanner class in the bluetooth package. One way to do this instead of trying to reconnect often would be to keep a list of nearby bluetooth units, and only connect when the service knows that the device is in range. Using BluetoothScanner to do this should be relatively simple, although one has to be aware that both the scanner and the service are threaded, which could cause some synchronization issues.

Every connected device is polled by the service for data on every iteration of the loop. This means that the service thread indirectly runs all the device code. An earlier design decision that was later reverted was to make all the devices threaded, but the synchronization issues this caused did not seem to be worth the performance gain, so this was scrapped. We are no longer sure if this is the case,

[2]This step is only applicable when the chest unit is activated.

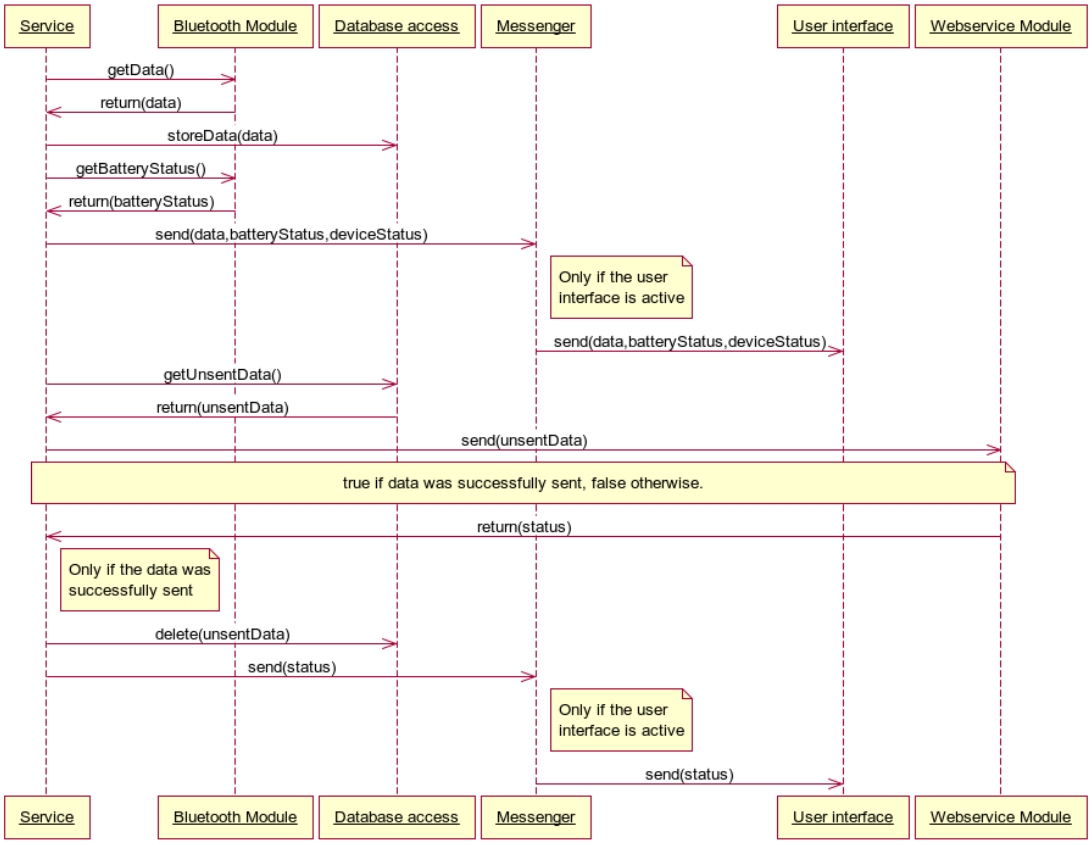

Figure B.1: The service cycle represented as a sequence diagram.

and it may be that threading the devices could cause significant performance gains. The service saves all the data it obtains from devices, so that they do not need to be using the datastore directly, before it passes said data on to the graphical user interface via a messenger object. Every sendInterval seconds, it will fetch unsent data from the datastore, and ask the webservice module to pass it on to the ESUMS server.

A rather large portion of the code in the service is there to facilitate easier addition of new devices to the code base. Although the architecture does not transparently allow for plugins as such, the createParser method puts the mechanics that have to be in the service in place for this, so that it could be performed at a later date. In combination with the initiateDevices method this could be used to allow for transparent addition of devices. Nearly all other code in the service is communication code that deals with updating the graphical user interface as necessary.

B.3.3 Web services module

The web services module functions as a data transmitter for the service. It doesn't ever directly use the datastore, it only receives data, processes it, and lets the caller know how it went. Originally, it was intended to have an interface consisting of three very simple methods:

- connect() - connect the webservice
- isConnected() - check whether the webservice is connected
- sendData(data) - this returns data that was transmitted

This had to be changed near the end of the project for a couple of reasons. The big one here is that the service can't really connect, and stay connected. A connection is made to obtain a security token that is used for future requests, but the webservice does not use a persistent streaming connection. What happens is that every time a batch of data has to be sent, a security token is obtained, and then it is used multiple times to call addMedicalDataPoint remotely, as defined by the MPOWER WSDL files. However, because this implicitly creates a HTTP connection, and blocks, this would stall the entire service until data transmission was done. Because of this, it had to be threaded, which changed the interface slightly.

- isTransmitting() - check whether a transmission thread is running
- transmitData(data) - start a data transmission
- getSentData() - get the data that was sent by the most recent transmission thread

Only one data transmission thread will ever run at the same time, and they are started periodically, if there is enough data stored on the handheld to justify a transmission. The procedure by the service is to check if there's a data transmission going every few minutes, if there is not, it fetches the data that was sent by last transmission, marks it as sent in the database, and then starts a transmission on data that hasn't been sent yet. The webservice then has no direct interaction with the data persistence layer, it is essentially a collection of data processing routines that are called from the application logic layer only.

Internally, the web services module uses a version of the ksoap2 library written specifically for android. Only a very small number of classes and methods here are used, due to the low requirements the module has to SOAP (Only two remote methods have to be called). SoapObjects are created manually, instead of serialized/deserialized to and from XML. This is not the recommended way to use

SOAP, but the overhead of doing it the way we did it is smaller, due to the low amount of functionality we required from the MPOWER webserver.

Because the MPOWER webserver currently does not support storing data in batches, we needed to store data points one by one, which is a pretty significant performance bottleneck. Work is in progress to rectify this issue serverside, after which a modification of the handheld code would be required to take advantage of the batch-storing method.

B.4 User interface layer

Part of the requirements of the customer were that the application should show data for users, if they were interested. This meant we had to add a user interface, and supporting code for it. The UI layer serves this purpose, and also allows for configuration of the application. Without this we would have to create a custom configuration format, and document it for nurses which had to use it.

B.4.1 User interface

The UI was intended to be as simple as possible, which meant we had to add custom widgets, specially designed for our application. The most important one was the DataView, which is the primary method of showing data in the application. It comprises several other widgets, some standard, and some custom, in order to display as much data as possible, in the limited screen real-estate available. This was quite a bit of work, as custom widgets aren't serialized automatically, the way standard widgets are, which again meant we couldn't define them using XML. This is the standard in Android, and saves a lot of work.

The UI consists of two Activities, which is Androids name for a single screen of information. The low number is due to our desire to keep the application as minimal and user-friendly as possible. Each UI has its layout defined in XML-files. These can only store the data available to us at compile-time, which means device-specific widgets have to be added at runtime, in code. For this reason the UI-layouts are mostly laid out in a logical form, keeping sections of the layout available for the code to add widgets.

User events in Android are handled in much the same way as any java UI-library does it. Widgets have *onEventHappened()* methods, which are private inner classes implementing a callback-function. This may not be the most elegant solution to UI coding, but is the one available to us. The application has quite a few of these, grouped together with the widget which uses it. We felt this was the most natural solution to the problem at hand, as it keeps related features of the

UI tightly coupled, and exposes the minimum amount of classes to the rest of the code.

Further, due to Androids restrictions on changing widgets from only the UI thread, we had to implement a custom solution for communicating with the UI. This involves a lot of boiler- late code, a single method requires three methods to implement.

We solved this problem by including a class ActivityServiceMessenger, which handles communication. Any interfacing between the UI and the rest of the application is handled in this class. It has methods for reporting status messages, displaying new data, and communicating settings between the components. Unfortunately, this creates a tight coupling between the messenger and any Activity which uses it. The most elegant solution to this would be to inherit any Activity from a common base class which implements all the common communication, and override messages which require special treatment in each Activity as needed. Java however, does not support multiple inheritance, and each Activity must inherit from Androids base Activity class or any of its descendants.

The problem only arises as the service has to initiate communication to the UI, and this is only necessary in the main activity. Because of this we decided to leave the coupling, and live with the restrictions imposed by this. If any other part of the UI should need to receive communications from the service, as would be the case of a detail-view for the chest- unit this problem would have to be solved. One possible solution might be to use an abstract class which inherits from Activity, and then let each application-specific activity inherit from this. The abstract class could then implement the common functionality. This solution has not been explored, but seems to offer the best tradeoff of features and complexity in java.

B.5 Known issues with application code

B.5.1 Unsolved bugs

1. Can't create bluetooth connection to peripherals on Samsung Galaxy. UUID of peripheral bluetooth services are required for this to work.

2. Periodic performance issue while transmitting data and reading from Nonin at the same time.

3. Android does not report to the application that bluetooth sockets close, reconnecting them does not currently work well. As a consequence of this, user can not seamlessly move in and out of range of handheld while taking measurements.

4. Failing to install devices into the datastore prior to storing data points leads to an undesirable situation where data points can be saved, but not retrieved.

B.5.2 Suggestions for future improvements

1. Make architecture fully support plugin peripheral devices.
2. Domain / device specific model is currently not isolated into one place of the code. Making sure to put all such code in one place would be good.
3. More informative error messages. Some errors are currently not reported.
4. Move more options into configuration (A lot of static final Strings).
5. Make bluetooth devices fully threaded.
6. Modify datastore to truly use batch storing and updating.
7. Modify datastore to allow DataPoints to be related to one another (Ie. measurement relation to measurement confidence).
8. Fetch and use more information about DataTypes from MPOWER webserver.
9. Graphical user interface should group data views on DataSource, so the user can easily see which data that comes from which device.
10. Batch-storing to webserver, when MPOWER supports this on serverside.
11. Implement enabling and disabling of connections to specific devices, instead of all devices.
12. Currently the application requires server address specification on the form *http://address/* with no flexibility. Adding some more clever parsing on this would be nice for the end user, ie. adding protocol and trailing slash if not present.

B.5.3 Unimplemented features

1. Graphical user interface does not support detailed view of data, magnifying of graphs and so on.
2. Battery state of Chest Unit is currently not implemented, as the circuits for correctly reporting the charge level are not in place yet.

3. Do something sane with Status report from MPOWER webserver in the webservice package. These are currently only read as success or failure, and the specific reasons for failure are ignored.

Appendix C

Test plan

C.1 Unit Testing

Initially, the plan was to write specific unit tests for all the classes and methods where it seemed necessary. However, this was never fully realized, and most of the testing and debugging of isolated classes and corresponding methods was done manually during implementation by the one responsible for that section of code. This approach is not very structured and does not guarantee a sufficient result. Still, for internal testing during implementation it worked fine. Due to the application being very dependent on external modules, creating proper test cases would also involve a lot of work simulating input and output, especially for the bluetooth and web service modules. Therefore manual testing and debugging was deemed sufficient during implementation, and structured testing could be performed at the end of the implementation phase, with the integration tests and complete system tests.

C.2 Integration Testing

Integration testing occurred toward the end of the project, when each module had been implemented. This testing was mostly done manually after connecting everything. The application was run and debugged step-wise, with two modules integrated with each other and tested for correctness. This was done with all the modules before integrating all of them into the complete application. When this was done, system testing and system integration testing was performed.

C.3 System Testing/System Integration Testing

This section describes the high level tests performed to make sure the system functioned as specified by the functional requirements. Some non-functional requirements were also included where applicable. Due to the acceptance tests covering the functional requirements, these tests were mapped to the corresponding backlog tasks. It was done this way to have a wider range of tests and limit test duplication. Together the tests should cover all the requirements and the desired functionality of the finished system. The tests include the results for each test as well as a comment section. Originally the intention was for the test plan to be seperated between system testing and system integration testing. However, our application has no real functionality without being connected to a bluetooth device and/or the ESUMS server. Testing with real input and output makes for a more realistic approach. Combining the two results in a much clearer test plan.

As table C.1 shows, most of the tests were performed successfully. The missing functionality was related to updating of GUI elements. ST06 could not be properly tested. However, all the requirements needed to perform the main tasks of the product, were fulfilled. Details can be found under each test.

Table C.1: System test result

System test	**Result**
ST01	Mostly complete
ST02	Complete
ST03	Complete
ST04	Complete
ST05	Complete
ST06	-
ST07	Complete

Test ID:	ST01
Description:	Testing of GUI elements
Backlog items tested:	S7, S8, S9, S10, S14, UI2, UI3, UI4, UI5, UI6, UI7, UI8, UI9, UI10, BT10
Backlog items not tested:	S1, S2, UI1
Backlog items:	Tests:
UI2, UI4, UI8, UI9, UI10	1. GUI displays data received from bluetooh device
UI3	2. GUI displays connection status with ESUMS server
S14, UI3	3. GUI displays connection status with bluetooth devices
S7, S8, S10, UI3, BT10	4. GUI displays battery status of bluetooth devices
UI5	5. GUI displays list of all available bluetooth sensors
UI6	6. Configuration file details can be changed from the GUI
S7, S8, S9, S10, UI7	7. Clicking status icons in the GUI gives the status of the selected item

Result:	Mostly complete
Comments:	1. GUI displays incoming data in real time in dynamic graphs. Threshold added, but is constant for all users. Fullscreen view not implemented 2. Server connection status displayed 3. Bluetooth device connection status displayed, but not updated if connection fails 4. Current bluetooth devices do not broadcast battery status 5. View of pre-configured devices exist in Configuration view 6. Configuration view present 7. Clicking status icons is used for refreshing connections

Test ID:	ST02
Description:	Handheld application searches for and connects to paired bluetooth devices
Backlog items tested:	S12, S13, BT1, BT2, BT3, BT4, BT5, BT6, BT7, DB6, BT12
Backlog items not tested:	
Backlog items:	Tests:
S13, BT1, BT2, BT3, BT4 S12, S13, BT5, BT6, BT7, BT12, DB6	1. Handheld automatically searches for nearby bluetooth devices when turned on 2. Handheld automatically connects to found bluetooth devices that match the settings from the configuration file
Result:	Complete
Comments:	1. Handheld searches for bluetooth devices 2. Pre-configured Nonin and chest unit automatically connected

Test ID:	ST03
Description:	Handheld application receives data from connected bluetooth devices
Backlog items tested:	S6, S15, BT8, BT9, BT11, DB1, DB3, P2, P4
Backlog items not tested:	
Backlog items:	Tests:
BT8, BT9 BT11, P2, P4 S6, S15, DB1, DB3, P4	1. Handheld requests bluetooth device to starts transmitting data 2. Handheld receives data 3. Handheld stores data in database
Result:	Complete
Comments:	1-2. Handheld automatically receives bluetooth data when a connection has been established 3. Incoming data is stored in database before it is sent to the GUI

Test ID:	ST04
Description:	Handheld application connects to ESUMS server
Backlog items tested:	S4, S11, W1, W2, W3, W4, W5, W6
Backlog items not tested:	
Backlog items:	Tests:
S11, W1, W2, W3, W4 S4, W5, W6	1. Handheld connects to ESUMS server automatically 2. Handheld authorizes as user according to configuration file
Result:	Complete
Comments:	1. Server connection attempted and performed at regular intervals while unsent data exists 2. Credentials specified in configuration file used for authorization

Test ID:	ST05
Description:	Handheld application transmits data to ESUMS server and handle responses
Backlog items tested:	S3, S5, W7, W8, W9, DB2, DB4, P3, P4
Backlog items not tested:	
Backlog items:	Tests:
DB2 S3, W7, W9, P3, P4 S5, W8, W9 S5, DB4	1. Handheld retrieves unsent data from database 2. Handheld parses data and sends to ESUMS server 3. Handheld receives response from server that data has been received 4. Handheld removes sent datapoints from database
Result:	Complete
Comments:	1. Unsent data easily retrieved 2. Unsent data sent to web module, where it is serialized and transmitted 3. Data received by server marked as sent 4. Cache of sent datapoints read and contents deleted

Test ID:	ST06
Description:	Adding additional external sensors
Requirements tested:	M1
Requirements not tested:	
Requirements:	Tests:
M1	1. Adding a new external sensor takes no more then 6 hours
Result:	-
Comments:	1. Bluetooth module highly extensible. Adding external sensors should not be a time consuming task, but has not been thoroughly tested.

Test ID:	ST07
Description:	Preventing data loss and maximizing uptime
Requirements tested:	A1
Requirements not tested:	
Requirements:	Tests:
A1 A1	1. Data received from bluetooth devices is stored in the database immidiately 2. Data retrieved from the database is not removed from database until confirmation is received from ESUMS server
Result:	Complete
Comments:	1. Incoming data is sent to database immidiately 2. No data is deleted unless marked sent by web service module

C.4 Acceptance Testing

Acceptance testing is meant to be performed in close contact with the customer and is supposed to cover what they expect from the finished product. In discussion with the customer, it was decided to base the acceptance testing on the requirements derived from the documentation supplied by the customer. This is basically the described functional requirements. Only the requirements relevant to our application were tested. The idea was basically to go through the requirements list one by one and ensure the system functioned accordingly. The details for each requirement can be found in the ESUMS original requirements document[4].

Table C.2 displays each requirement ID, along with status and any comments. As the table shows, most of the desired requirements were implemented successfully. REQ079 not being implemented was due to necessary data not being available at the time. The partial completion of REQ121 was because of holes in the Android bluetooth API. Details can be found under each test. Overall though, ESUMSDroid passed acceptance testing according to the chosen criteria. The missing requirements were not crucial to the main functionality of the product, as they relate mostly to GUI notifications and the updating of these.

Table C.2: Acceptance test result

ID	Status	Comments
REQ024	Complete	
REQ122	Complete	
REQ121	Partially complete	Limitations in Android
REQ077	Complete	
REQ079	Not implemented	See REQ121
REQ021	Complete	
REQ023	Complete	
REQ073	Complete	
REQ002	Complete	

- **REQ024 - Handheld vital signs application**
  Status: Complete
  Prerequisites:
  - The application is installed
  - Bluetooth is enabled
  - Device is paired with external bluetooth devices
  - Application is properly configured for devices

  Steps:
  - Start the application
  - Wait for the device connection status to turn into a green arrow.
  - Assert data arrives from devices.

  Comment:

- **REQ122 - Attachment of external sensors**
  Status: Complete
  Prerequisites:
  - The application is installed
  - Bluetooth is enabled
  - Device is paired with Nonin

  Steps:
  - Start the application
  - Wait for the device connection status to turn into a green arrow.

  Comment:

- **REQ121 - Handheld system management application**
  Status: Partially complete
  Prerequisites:
  - The application is installed
  - Bluetooth is enabled
  - Device is paired with external bluetooth devices
  - External device is running

Steps:

- Start the application
- The top row of the application shows the status of connections.

Comment: This requirement is only partially completed, as Android never reports a bluetooth connection as closed. Because of this, the application does not report a device leaving range or failing. This is a limitation of Android. If a device fails, it is possible to manually reconnect, by pressing the status icon twice.

- **REQ077 - Handheld handling of failing communication with server**
  Status: Complete
  Prerequisites:
  - The application is installed
  - Application is configured for server.
  - Server is not running

  Steps:
  - Start the application
  - The application immediately tries to connect to server. Wait for icon to turn into a red X.
  - Start the server
  - The application retries after a maximum of 60 seconds, and completes.

  Comment:

- **REQ079 -**
  Status: Not implemented
  Prerequisites:
  - None

  Steps:
  - None

  Comment: The chest unit does not currently implement battery readings, and was therefore not implemented. The remainder of this requirement is handled by REQ121

- **REQ024 - Handheld vital signs application**
  Status: Complete
  Prerequisites:
  - The application is installed
  - Bluetooth is enabled
  - Device is paired with external bluetooth devices

  Steps:
  - Start the application
  - Wait for the device connection status to turn into a green arrow.
  - Assert data arrives from devices.

  Comment:

- **REQ021 - Continuous measurement view**
  Status: Complete
  Prerequisites:
  - Same as REQ024

  Steps:
  - Same as REQ024

  Comment: As the chest unit currently does not support batch data mode, and the Nonin does not support it at all, this requirement is a duplicate of REQ024

- **REQ023 - Chest unit standard communication protocol**
  Status: Complete
  Prerequisites:
  - None

  Steps:
  - None

  Comment: The application communicated correctly with the chest unit, as specified in the document ESUMSkommprot.doc. Bluetooth has encryption of data enabled as standard. The devices only allow pairing with a single device at the time.

- **REQ073 - Handheld vital signs application**
  Status: Complete
  Prerequisites:
  - Application is configured for no devices

  Steps:
  - Start application.
  - Assert the application does not attempt to connect. The status row shows a red X.

  Comment: Multiple chest unit operation was not tested, as we only had one available. It is assumed that bluetooth can handle this. Both Nonin and Chest Unit were operated simultaneously.

- **REQ003 - Identification of corresponding services**
  Status: Complete
  Prerequisites:
  - The application is installed
  - Server is running

  Steps:
  - The application immediately tries to send data.
  - The application authorizes with the server in the background
  - If the authorization details are not valid, the application receives no notification from the server, and handles this as a generic communication failure.

  Comment: The server has no useful notification of failed logins, due to wrong username or password, and cannot be handled specially by the application.